Aalto University

School of Science

Master's Programme in Computer, Communication and Information Sciences

Kristian Klemets

Forecasting Hourly Parking Occupancy with Multiple Seasonalities

Master’s Thesis

Espoo, June 8, 2020

Supervisor: Professor Alexander Jung

Thesis advisor(s): Roope Tervo, M.Sc.

Aalto University
School of Science
Master's Programme in Computer, Communication and Information Sciences

ABSTRACT OF MASTER´S THESIS

Author:
Kristian Klemets

Title:
Forecasting Hourly Parking Occupancy with Multiple Seasonalities

Date:	Pages:
June 8, 2020	60
Major:	Code:
Computer Science	SCI3042

Supervisor:
Professor Alexander Jung

Advisor:
Roope Tervo, M.Sc.

Forecasting parking occupancy in city areas has become increasingly important to give the city and drivers a way to predict the available parking spaces. The city can use this information for planning and the drivers can predict where to park their car and avoiding the need of searching for a parking space.

In this paper we introduce various prediction models for forecasting parking occupancy on an hourly level and compare their forecasting performance with a dataset of parking instances. The tested models include linear regression, gradient boosting, SARIMAX, TBATS, Facebook Prophet, and two neural network classes: long short-term memory and gated recurrent unit.

The experimental model results were compared against each other, and the evaluated results suggest that gradient boosting is the best performing model for our dataset. The results are evaluated both in the error metrics and training times of the models.

Keywords:
Machine learning, time series, gradient boosting, FBProphet, TBATS, neural networks, long short-term memory, gated recurrent unit, parking prediction

Language:
English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tieto-, tietoliikenne- ja informaatiotekniikan maisteriohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä: Kristian Klemets	
Työn nimi: Parkkeerausten määrän ennustaminen tunneittain kausittaisesta datasta	
Päiväys: 8. kesäkuuta 2020	Sivumäärä: 60
Pääaine: Tietotekniikka	Koodi: SCI3042
Valvoja: Professori Alexander Jung	
Ohjaaja: Diplomi-insinööri Roope Tervo	

Parkkipaikkojen käyttöasteen ennustaminen kaupunkialueilla on tullut yhä tärkeämmäksi, jotta kaupungilla ja kuljettajilla on tapa ennakoida vapaana olevia pysäköintipaikkoja. Kaupunki voi käyttää ennusteita liikenteen suunnitteluun ja kuljettajat voivat ennakoida, mihin pysäköidä autonsa ja välttää pysäköintipaikan etsimisen tuomia haittapuolia, kuten bensan- ja ajankulutusta.

Tässä työssä esittelemme erilaisia ennustemalleja pysäköintien käyttöasteen ennustamiseksi tunnin välein ja vertaamme niiden ennustekykyä pysäköintitapahtuma tietoaineistoa käyttäen. Testattuihin malleihin sisältyvät lineaarinen regressio, gradient boosting, SARIMAX, TBATS, Facebook Prophet sekä kaksi neuroverkkoluokkaa: long short-term memory ja gated recurrent unit.

Mallien alustavat tulokset viittaavat siihen, että gradient boosting antaa parhaat tulokset työn aineistoa käytettäessä. Mallien vertailun perusteena käytettiin sekä suorituskykyä, että koulutusaikoja.

Avainsanat: Koneoppiminen, aikasarjat, neuroverkot, parkkeerausten ennustus	
Kieli: Englanti	

Acknowledgements

I wish to thank a few individuals that helped me throughout this project: Thank you, Alexander Jung, for supervising my thesis and hosting group meetings with interesting topics, where we could also seek for advice. Thank you, Mikko Kivistö and Jaakko Partanen, for helping and making this project possible. Thank you, Aslan Venejoki, for suggesting the topic and helping with the planning of the project. Thank you, Tuomas Suutari, for help with planning and reviewing the work. Thank you, Roope Tervo, for working as the thesis instructor. Huge help with the planning, structuring, and proof reading of both the implementation and the written work.

I want to thank Anders Innovations and City of Helsinki for giving me this opportunity to research an interesting topic and providing resources for the experiments and research.

Espoo, June 8, 2020

Kristian Klemets

Abbreviations and Acronyms

ARMA	Autoregressive moving average
ARIMA	Autoregressive integrated moving average
SARIMA	Seasonal autoregressive integrated moving average
SARIMAX	Seasonal autoregressive integrated moving average with exogenous variables
OLS	Ordinary least squares
MAE	Mean absolute error
RMSE	Root mean square error
MAPE	Mean absolute percentage error
MSE	Mean squared error
ANN	Artificial neural network
RNN	Recurrent neural network
LSTM	Long short-term memory
GRU	Gated recurrent unit
TBATS	Trigonometric Box-Cox transform, ARMA errors, Trend, and Seasonal components
API	Application programming interface
i.i.d.	Independent and identically distributed

Contents

1. Introduction

The number of cars in cities is increasing constantly and this introduces a problem of depleting parking spaces. A lot of the traffic in cities, 30% by some estimations, is caused by cars that are searching for parking spaces [1]. Drivers might even cause safety hazards by getting distracted or not paying attention to the traffic around them when searching for an open parking spot. Also, extra fuel is consumed if the parking spot is not found in a reasonable time.

Cities often have parking monitoring systems in place which is used to control the parking space usage throughout the city. The monitoring systems include parking meters on streets, parking halls and private parking operators. Data generated from the monitoring systems can be gathered to a centralized database with the location and timestamps included for data processing use. With all this groundwork in place, we can use the data to infer the parking occupancy of different parking areas around the city. From the parking occupancy data, we can infer parking habits and it can be used to plan out the parking infrastructure to better meet the need of the users, the drivers. We can also extract a time series from the gathered parking data to create future predictions for parking occupancy. All the gathered data can be used to help in future planning of traffic and parking monitoring.

This work is based on a customer project for the city of Helsinki. They had a large dataset of parking data from different traffic operators that could be used for learning parking patterns based on location and time. They wanted to be able to predict the parking behavior of drivers for a week in advance. The goal of this paper is to create a prediction model for forecasting the parking space availability by parking locations on an hourly basis. As the (prototype) product we provided a prediction of parking space availability around Helsinki city. In practice we developed an API that provides an hourly forecast of available parking spaces by regions. The availability is indicated by a percentage or number that can be visualized as a heat map for example.

Existing work in the field utilize pre-installed sensors and on-vehicle approaches specialized for the purpose of parking prediction. This approach somewhat differs from our approach since the data we use is gathered as part of the existing parking monitoring system from different parking operators. This data is then re-utilized to fit the purpose of parking predictions. Our application can be implemented without any extra resources if a city has similar parking monitoring systems in place and the data from different sources are gathered to a central database.

1.1. Basics of Machine Learning

Machine learning problems can roughly be divided into supervised and unsupervised methods. In this paper we focus on the supervised problems.

In supervised machine learning, the data has two parts: features and labels. Features consist of measurements that describe a data point. In general, features can be easily measured or computed automatically from the data. Feature space χ is the set of all possible values a feature vector can have. Labels are properties of the data of interest. On the contrary to features, labels cannot be easily measured or computed automatically. Constructing a label set usually requires manual work of domain experts. The range of all possible values a label can take is called the label space Υ. [2]

In machine learning systems, it can be assumed that data labels are related to features by some unknown process y = f(x), where y is the target and x is the feature vector. The objective of supervised machine learning is to create a hypothesis map h: $\chi \rightarrow \Upsilon$ that maps the feature vector to a predicted label $y = h(x)$, so that the true signal of the process f(x) is approximated as closely as possible. The approximation is done by using an error measure, L((x, y), h), called the loss function. In modern machine learning problems, the challenge often comes from the high dimensionality of the data. Moreover, highly non-linear predictors are used in many machine learning methods that are computationally demanding to fit to the given data points. [2]

In order to make predictions for new data, a hypothesis must be learned from existing training data. This hypothesis is part of a hypothesis space, that is limited to a set of parameters, also called a model. The goal of training the model is to find an optimal set of parameters, or in other words, optimizing the hypothesis. However, first the training data needs to be split into training, validation, and test sets. Parameters θ of the model $h(x; \theta)$ can be estimated using the training set. Additionally, some models have supplementary hyperparameters that must be set manually before training the model. In order to optimize the hyperparameters of the model, validation set can be used. Test set is used for measuring the performance of the model after the training has been done. The division of training, validation and test sets is usually 50% / 25% / 25% of the training data, respectively.

1.1.1. Time Series Forecasting

A dataset D that contains historical measurements x_t that are observed at time intervals t is called a time series [3]. Time series are used for multiple purposes in science and different industries. Applications include forecasting of weather, financial statements and electricity consumption. [4]

The measurements in a time series may be continuous or measured at discrete times throughout the entire time index. These time series are called continuous and discrete time series, respectively. In this paper we focus on the discrete time series. A discrete time series is often sampled at equal intervals to have consistency in the data.

Before modeling and forecasting a time series, it is important to analyze the data first to find distinct properties of the data that might be important in the modeling process. First it is recommended to do some preliminary work to clean the data from distinct errors and outliers. After cleaning the data, we can recognize certain components by taking a decomposition of the data. Example of a decomposition is plotted in Figure 1.1.1. These components include trend and seasonal variation of the time series. In the data, trend can be observed as a steady upward growth or downward decline in the time series. Seasonal variation appears as measured values correlating to each other every t steps, where t is for example a week or a year. Correlation between successive measurements in a time series is often called autocorrelation. It can be used to spot the correlations in a time series with various *lags*. Lag is essentially a delay in time series. For example, autocorrelation at lag t is the correlation between values that are t periods apart from each other. The residual plot in Figure 1.1.1 shows how the time series behaves after removing the trend and seasonality from the original time series data. [5]

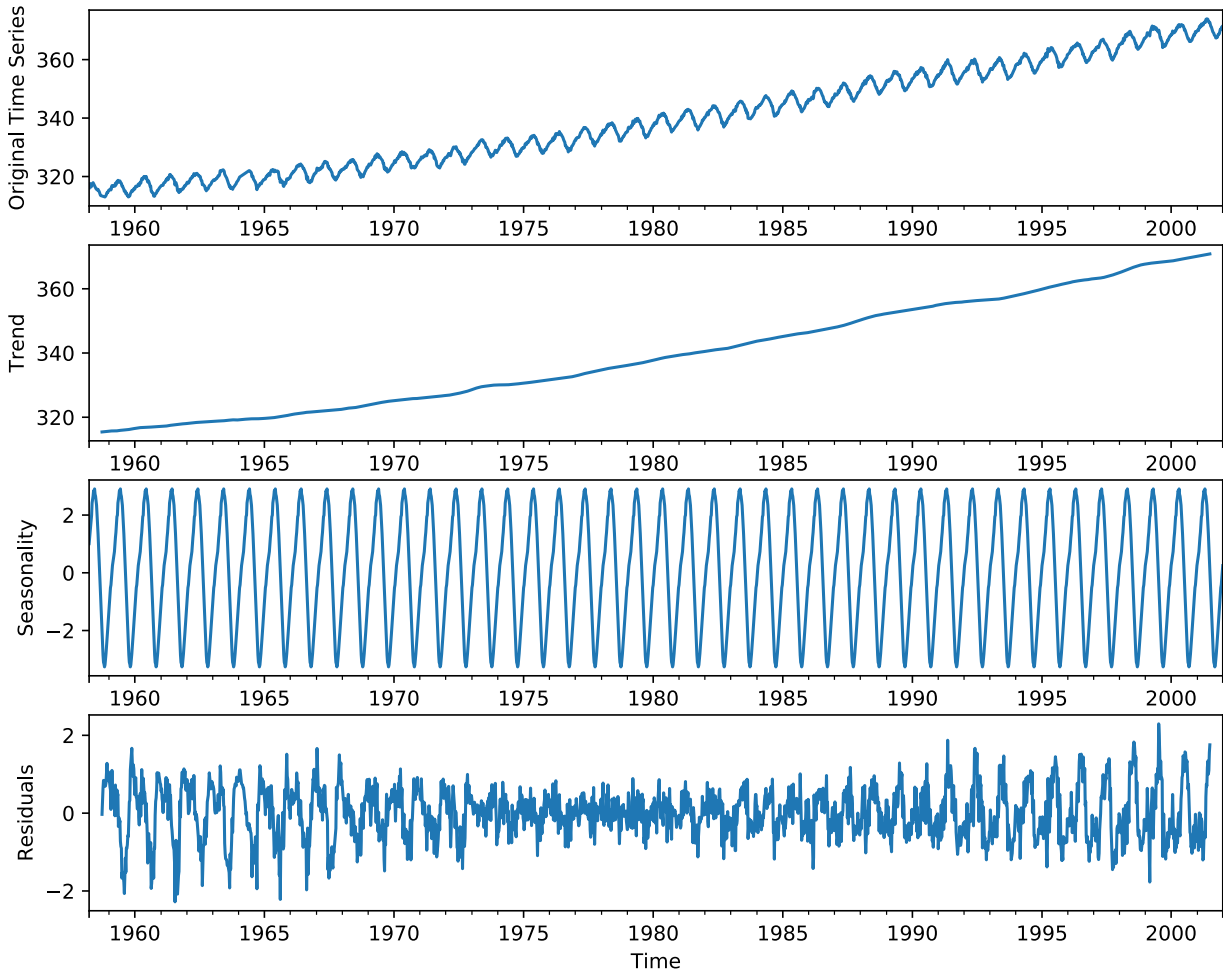

Figure 1.1.1: Example plot of time series decomposition.

1.2. Thesis Structure

Chapter 2 examines past scientific approaches on similar problems that is described in this paper. Chapter 3 takes a closer look on how the dataset is structured and analyzes the form of the data. Chapter 4 analyzes the theory behind the methods that are later used in this paper to create the predictions on the parking dataset. Chapter 5 shows the process of implementing the chosen prediction models and creating the experiments for them. Chapter 6 states the resulting metrics obtained from the experiments and supplements them with a summary description. Chapter 7 discusses the different stages of the project in depth. Chapter 8 gives a conclusion to the paper describing what was achieved.

2. Related Work

Available parking space forecasting in large cities has been a known problem for many years [6]. Various data sources have been used in the past implementations of parking prediction systems. Often statistical methods and machine learning is used as tools to process the data.

Mobile phones, carried by vehicle drivers, have been used to collect data with the help of statistical methods and machine learning. Xu et al. [7] use various methods to detect the parking status of users and mobile phone GPS for tracking the location of parked vehicles. Their mobile application monitors the connections between a smart phone and an in-vehicle Bluetooth system. When the Bluetooth connection between a car and a cell phone gets disconnected, the application determines that the car has been parked. They also utilize a pay-by-phone parking application that communicates to their application when a parking event has started and when it ends. Lastly, they measure the accelerometer of the smart phone, and with the help of a classification model, determine when a user starts/stops driving and thus indicates a change in the parking status. Also, Nandugudi et al. [8] and Koster et al. [9] use the accelerometer for parking status detection in their work. Chen et al. [10] used fuzzy logic forecast models and location provided by mobile phone GPS to find the best parking location in park-and-ride facilities with public transport connections. Parking availability and travel time were used as parameters to find the optimal parking facility for users.

A network of vehicles (ParkNet) captured information about parking space availability in the work of Mathur et al. [11]. Vehicles equipped with GPS trackers and ultrasonic sensors were used to capture parking data while driving. The ultrasonic sensors were used to determine if a parking spot is free in the current location that the GPS tracker provides.

A real time on-street approach in Berlin predicts parking space occupancy by utilizing real time parking data from sensors mounted on walls and streetlights. They use a “Neural Gas” algorithm in combination with threaded history information captured by the sensors to predict the parking occupancy. [12] Another research project uses information provided by a parking ticket mobile application, number of parked vehicles, and traffic flow volumes to develop a real-time parking occupancy model. This model works without any roadside infrastructure and uses only existing real-time data sources in the city of Vienna. [13]

The previous examples focused mainly on on-street parking, but a few approaches focus on monitoring parking garages with different approaches and sensors. Badii et al. [14] utilize the

fact that parking garages are often already monitored with sensors, predetermined capacity, and a parking ticket system which makes it easier to gather accurate data. Their solution uses a Bayesian regularized neural network exploiting historical, traffic flow and weather condition data to predict available parking spots in monitored city parking garages.

Ionita et. al. [6] took the monitoring approach a bit further by creating the SFPark parking system in San Francisco. They collected parking data from parking meters from over a two-year period. They also collected data form traffic flows, events, fuel prices, and parking prices in different locations. All these factors can affect the location and volume of parked cars. They used multiple techniques, including support vector machines, multilayer perceptron, decision trees, and gradient boosting, to achieve good results in predicting parking occupancy around the city.

3. Dataset and Analysis

Our dataset consists of parking entries around the Helsinki city area. There are about 10 million data points of parking data available today and the amount is increasing every day. The parking data points are described in depth in Table 3.1. To refine the data into machine processable format, we must divide the dataset according to regions. Each region was parsed into hourly parking counts. This resulted in the dataset that is shown in Table 3.2.

Time series of parkings are usually represented as parking counts in manually selected geographic areas within a time step. The timestamps of parkings are recorded with a resolution of one second. However, for scientific purposes, too much granularity is often unnecessary and might even be costly in the creation and processing of the time series. Usually no drastic changes in parking counts happen within seconds or minutes, thus, an hourly precision is often enough.

Several different factors can affect the parking counts. For example, weather, season, events and holidays like Christmas or Midsummer have a clear impact on the number of parkings. These factors might cause non-linear effects when combined with the seasonalities of the data, creating unforeseen patterns on the parking counts, which makes it more difficult to make precise predictions based on the history of the data.

Attribute name	Attribute type	Attribute description
zone	int	Parking zone. City area is divided to 3 zones. This is populated in 100% of all parkings.
region	object	A related region object. The region has a GIS location area marked with multiple polygons. The regions are basically parts of town (Pasila etc.). This is populated in ~41.1% of all parkings.
parking area	object	A related parking area object. The area has a GIS location area marked with multiple polygons. The parking areas are basically parking lots or streets. This is populated in ~24% of all parkings.
terminal	object	A related terminal object. The terminal has a GIS location point as its attribute. Terminals are basically the parking terminals on streets. This is populated in ~19% of all parkings.
location	GIS location	GIS location. This is a very precise GPS coordinate point. This is populated in ~41.3% of all parkings.
start_time	datetime	Start time of the paid parking.
end_time	datetime	End time of the paid parking.

Table 3.1: Structure of a single data point of the initial dataset.

	Kaartinkaupunki	Etu-Töölö	Marjaniemi	...	Harju
2017-09-21 00:00:00	12	32	0	...	42
2017-09-21 01:00:00	12	32	0	...	43
...	...	...	...	...	...
2019-07-09 23:00:00	23	11	2	...	55

Table 3.2: Example of the hourly dataset divided to regions.

Before using a dataset, it is often wise to select and preprocess the data so that it has consistent variance and amplitude. This preformatting improves the predictive abilities of the model by reducing over- and underfitting.

In our parking dataset we have an interval in the beginning where the service was not in full use yet. During this interval, not all areas have data, and the parking numbers are much smaller compared to current situation, so these dates are cut off. Also, there are a couple of instances of extremely high peaks in parking numbers. The peaks are probably a cause of a correction in the system. Cut these outliers to avoid affecting the models with similar sudden changes. The aforementioned instances can be seen in Figure 3.1.

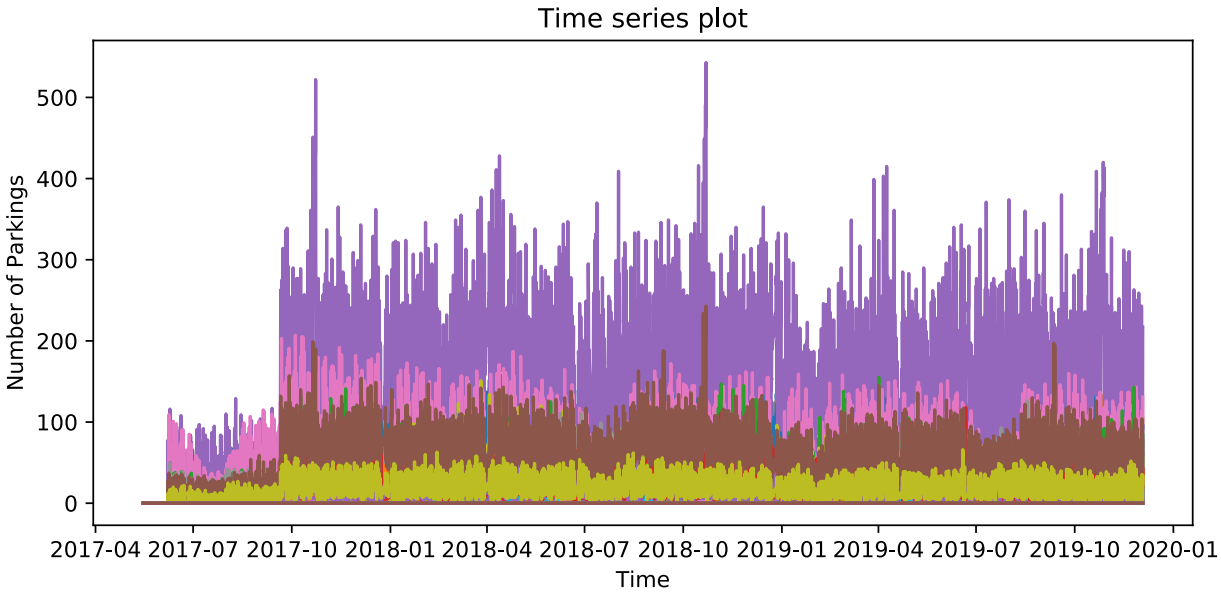

Figure 3.1: Parking data of all parking regions as a function of time.

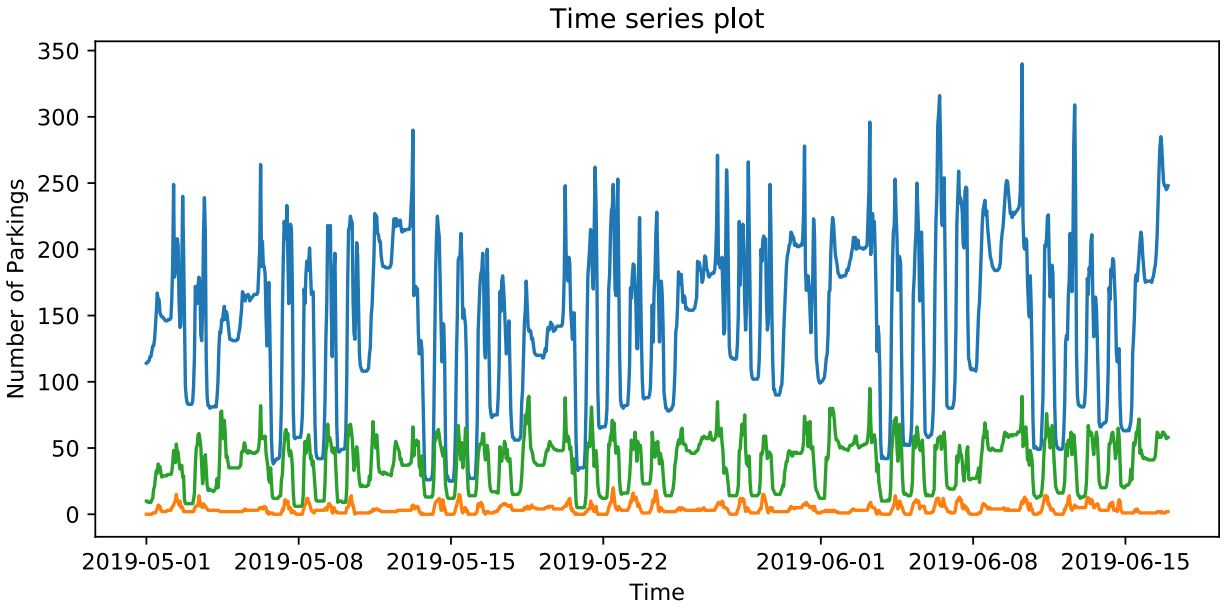

Figure 3.2: A closeup of the parking data of three very different parking regions as a function of time.

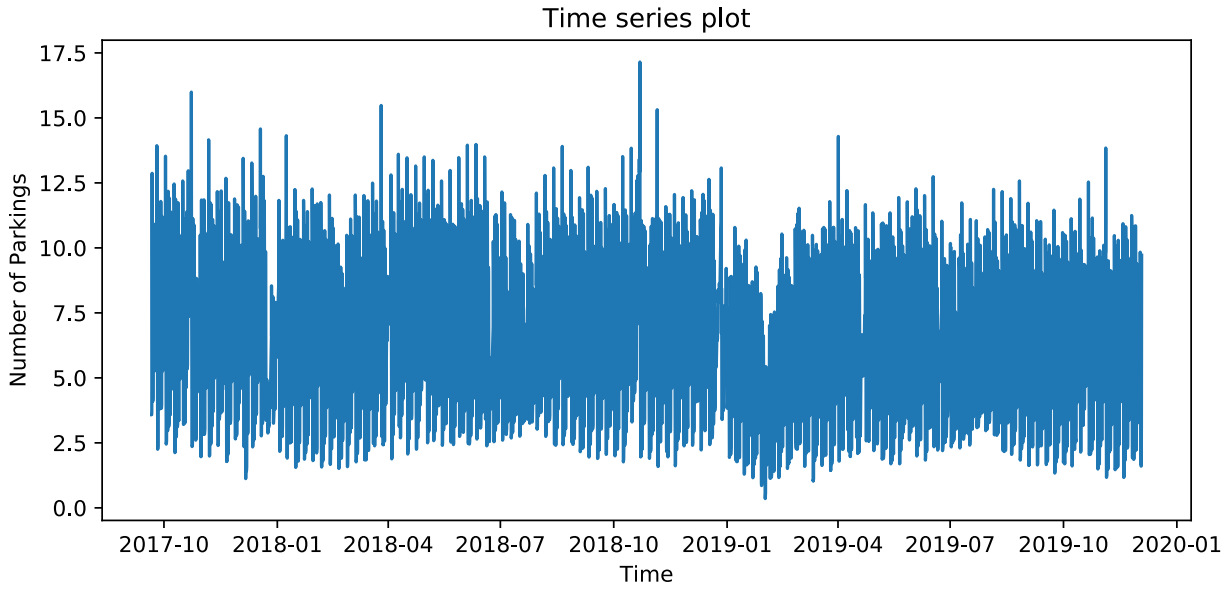

Figure 3.3: Time series plot of a single parking region as a function of time. The beginning interval and extreme values have been cut off.

Next, the data needs to be normalized. The normalization is beneficial because with the scaled features, we avoid large weight values and it makes the model training faster and less prone to blowing up.

By analyzing the data (Figure 3.1 – 3.4), we can see that there are multiple seasonalities involved in the dataset. There is the weekly cycle of weekends and daily night/day cycle. Also, the cycle of office hours on weekdays is apparent from the data. In addition, some effects caused by the time of year, are visible in the parking numbers. For some models, we need to add extra handling for these seasonalities. Additionally, holidays are an essential part of the problem because they are inconsistent. The holidays showed a clear correspondence to changes in the data. Usually a holiday is shown as a lowered amount of parkings in the dataset. We mark holidays by adding a new boolean feature that indicates if the timestamp of a datapoint is during a holiday. This way the model knows that data points during holidays behave differently from the normal cycle.

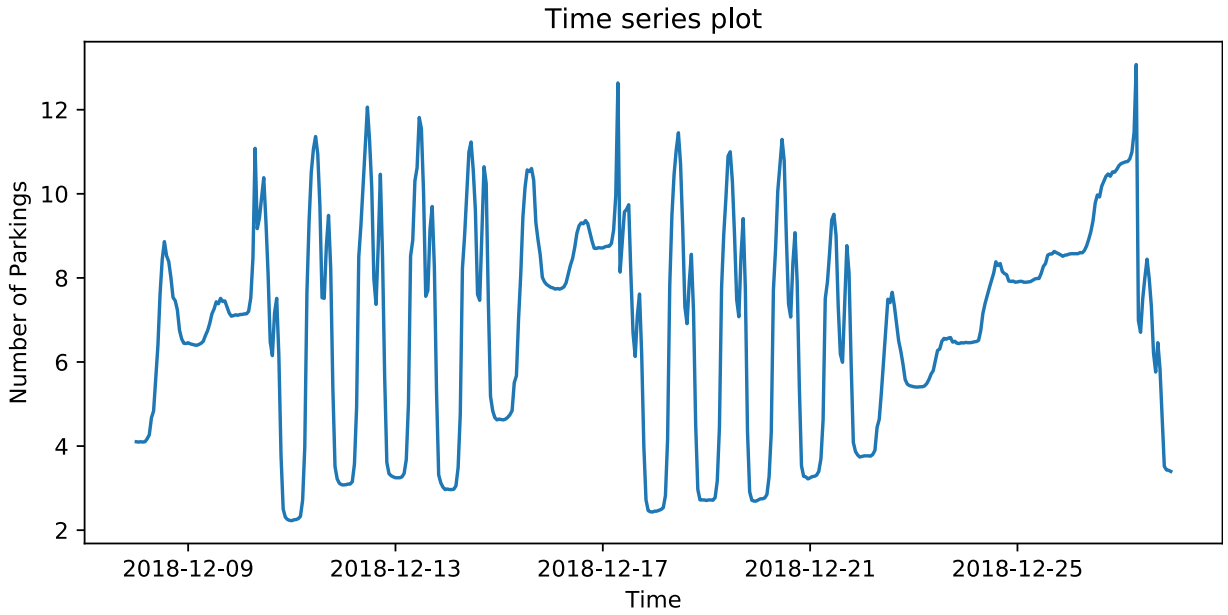

Figure 3.4: Close up of the time series plot of a single parking region as a function of time. Christmas holiday shows a distinct effect compared to the signal of regular weeks.

Autocorrelation tells us how the data correlates with the lagged values of the data. From Figure 3.5. we can see that there is high correlation with lags (24, 48, 72, …), meaning that there is a daily correlation in the data. Also, a higher peak can be seen on the lag 168, telling us that there is high correlation on the weekly level.

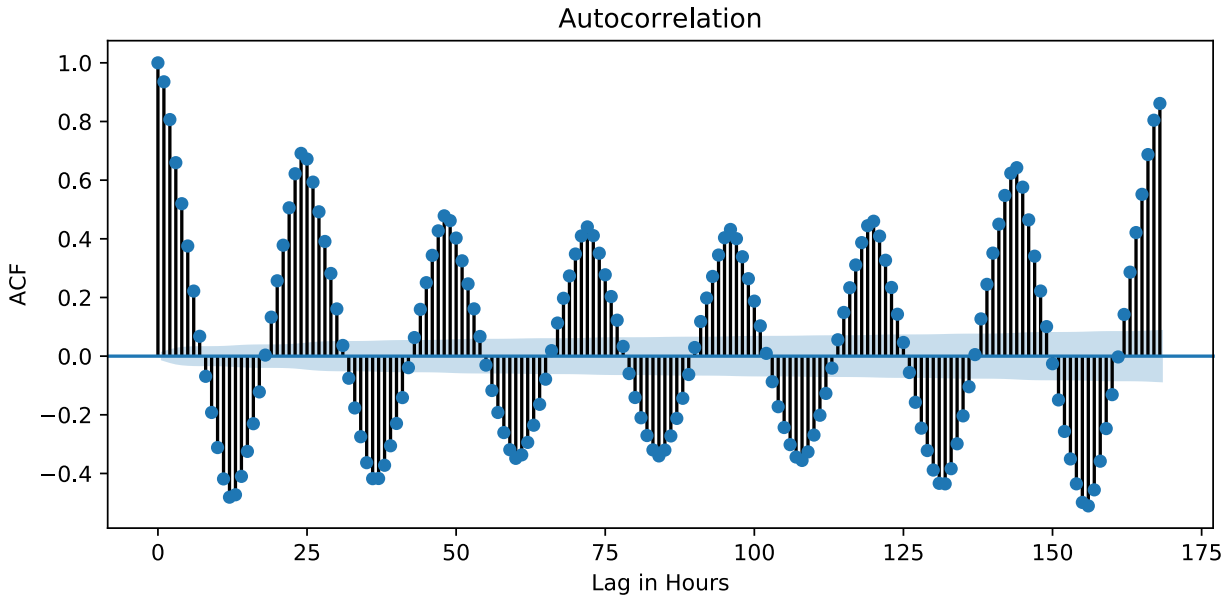

Figure 3.5: Autocorrelation of the mean of all parking regions. 95% confidence intervals are visualized as the blue cone. Correlation values that fall outside the cone are likely to be a correlation, rather than a statistical fluke.

Partial autocorrelation shows the correlation of the lags by removing the effect of intermediate values. This means that it can calculate the direct correlation between values x_t and x_{t-2} by removing the correlation of x_{t-1} from the formula, allowing us to see direct effects of each lag step. From Figure 3.6 we can see that the highest peaks are in the increments of 24 and some of the peaks are negative. Also, the weekly peak at 168 is not clearly higher as in the autocorrelation plot in Figure 3.5. This indicates that the daily seasonality (lag 24) overrules the weekly seasonality (lag 168).

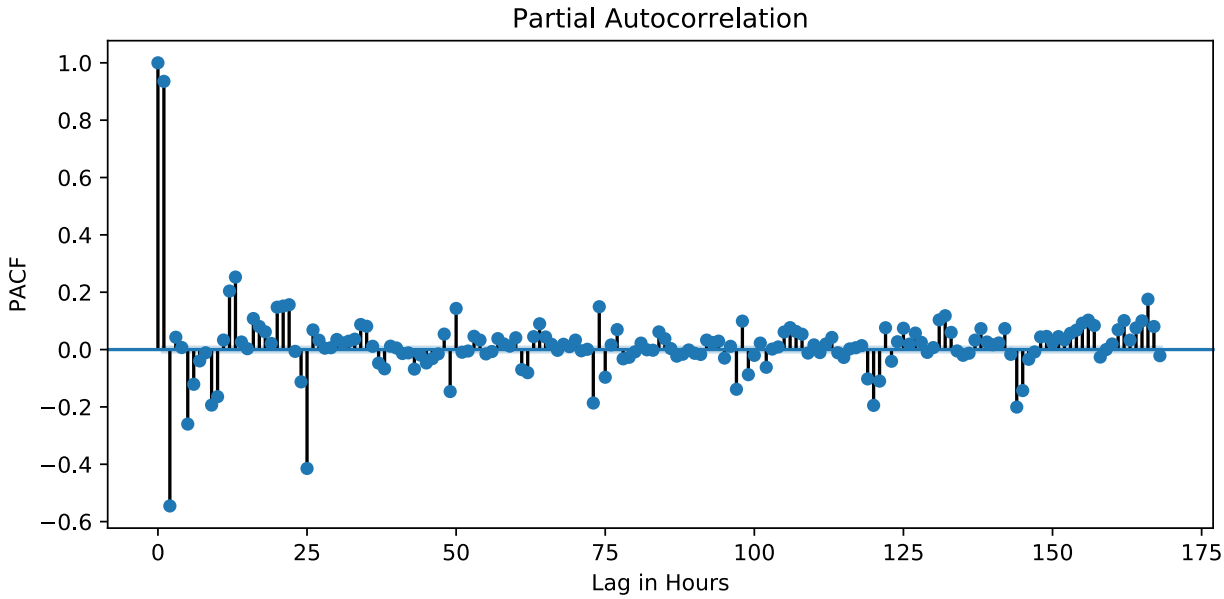

Figure 3.6: Partial Autocorrelation of the mean of all parking regions.

The curve in Figure 3.7 represents the Probability Density Function (PDF) of the time series. The density histogram is a representation of the time series where the number of parkings are grouped into bins with constant density and the y axis showing the frequency of the bin. From Figure 3.7 we can see the average range of values for the datapoints and thus the average scale of parking amounts for the regions.

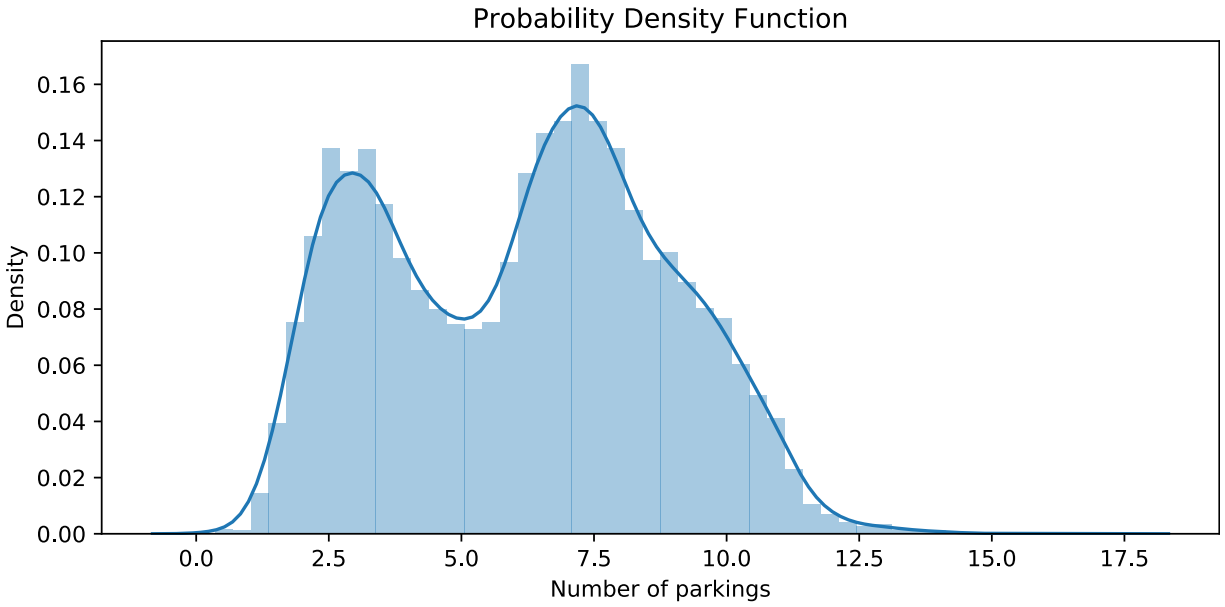

Figure 3.7: Probability Density Function of the mean of all parking regions.

4. Evaluated Methods

4.1. Linear Regression

In statistical analysis and machine learning, linear regression is one of the most commonly used models. It is an approach of modeling linear relationship between a scalar response and a set of explanatory variables. If there is only one explanatory variable, the process is called simple linear regression, while in case of multiple explanatory variables, the method is called multiple linear regression. [15]

In linear regression, relationships between the parameters are modeled using linear predictor functions. The unknown model parameters are estimated during the process. If we assume that the random label space $\boldsymbol{y} \in \mathbb{R}$ is related to feature space $\boldsymbol{x} \in \mathbb{R}^n$, we can create a predictor function $h^{(\boldsymbol{w})}$ to estimate this relation as follows:

$$\boldsymbol{y} \approx h^{(\boldsymbol{w})}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} \tag{4.1.1}$$

T denotes the matrix transpose. The goal is to estimate the weight vector $\boldsymbol{w} \in \mathbb{R}^n$ by minimizing the average squared error loss (mean squared error) between the estimated values of $h^{(\boldsymbol{w})}(\boldsymbol{x}^{(i)})$ and labels $y^{(i)}$. [2]

$$\begin{aligned} \boldsymbol{w}_{opt} &= \arg\min_{\mathbf{w}\in\mathbb{R}^n} \left(\frac{1}{m}\right) \sum_{i=1}^{m} \left(y^{(i)} - h\left(\boldsymbol{x}^{(i)}\right)\right)^2 \\ &= \arg\min_{\mathbf{w}\in\mathbb{R}^n} \left(\frac{1}{m}\right) \sum_{i=1}^{m} \left(y^{(i)} - \boldsymbol{w}^T \boldsymbol{x}^{(i)}\right)^2 \end{aligned} \tag{4.1.2}$$

In the parking dataset, the label space $\boldsymbol{y}$ would correspond to the number of parkings at time steps $i = 0 \ldots m$. The feature space $\boldsymbol{x}$ corresponds to the 6 features for the parking dataset at each time step i. Linear regression needs to be applied separately for each region in the dataset.

4.2. Gradient boosting

In this research we are looking into the CatBoost gradient boosting toolkit and the paper discussing the techniques behind it by Prokhorenkova et. al. [16] Gradient boosting is a prominent method in machine learning, that is used in multiple applications ranging from classification problems and recommendation systems [17] to weather forecasting [18]. Essentially, gradient boosting is a technique where ensemble predictors are constructed by performing gradient descent in a functional space.

Let us assume that we have a series of datapoints D = $\{(x_k, y_k)\}_{k=1\dots n}$, where $x_k = (x_k^1, \dots, x_k^m)$ is a feature vector of m features, corresponding to the 6 features of the parking dataset at time step k and $y_k \in \mathbb{R}$ is a numerical target response, corresponding to the number of parkings at time step k. Gradient boosting needs to be applied separately for each region of the parking dataset.

As an example, (x_k, y_k) are i.i.d. according to some unknown distribution P. The goal is to train a function $F: \mathbb{R}^m \to \mathbb{R}$ that aims to minimize the expected loss that is notated as $\mathcal{L}(F) \coloneqq \mathbb{E}L\big(y, F(x)\big)$. Here L is a smooth loss function and (x, y) are part of the dataset.

An iterative sequence of approximations $F^t: \mathbb{R}^m \to \mathbb{R}$, $t = (0,1, \dots)$ are built by a gradient boosting procedure. We get F^t from the previous approximation F^{t-1} in an additive manner: $F^t = F^{t-1} + \alpha h^t$, where α is a step size. The function $h^t: \mathbb{R}^m \to \mathbb{R}$ is chosen from a set of functions H:

$$\begin{aligned} h^t &= \arg\min_{h \in H} \mathcal{L}(F^{t-1} + h) \\ &= \arg\min_{h \in H} \mathbb{E}L(y, F^{t-1}(x) + h(x)) \end{aligned} \tag{4.2.1}$$

For the minimization problem, the Newton method is often utilized, using second-order approximation of $\mathcal{L}(F) \coloneqq (F^{t-1} + h^t)$ at F^{t-1} or taking a (negative) gradient step. The gradient step h^t is usually approximated as follows:

$$h^t = \arg\min_{h \in H} \mathbb{E}\big(-g^t(x, y) - h(x)\big)^2 \tag{4.2.2}$$

In this paper we use CatBoost library which is an implementation of the gradient boosting method in Python language. As base predictors CatBoost uses binary decision trees. To build a decision tree, a recursive partition of the feature space R^m must be split into several tree nodes

using the values of some splitting attributes a. The attributes a are usually binary variables that identifies if a feature x^k exceeds a threshold t:

$$\begin{aligned} a &= 1 \; if \; x^k > t, \\ else \; a &= 0 \end{aligned} \tag{4.2.3}$$

Each leaf of the tree gets a value assigned, which is an estimate of the response y. The decision tree can be written as:

$$\begin{aligned} h(x) &= \sum_{j=1}^{J} b_j a \\ a &= 1 \; if \; x \in R_j, \\ else \; a &= \; 0 \end{aligned} \tag{4.2.4}$$

where R_j is corresponding to the leaves of the tree. [16]

4.3. Autoregressive Moving Average

Autoregressive moving average or ARMA is a tool for forecasting future values in a time series X_t. ARMA can be divided into two parts, autoregressive AR and moving average MA. Both parts can be represented as its own model. AR aims to create a regression of its own lagged values. The MA term is a linear combination of past errors in the time series. The model is often notated as ARMA(p, q) where p is the AR term an q is MA term.

The autoregressive process AR(p) is dependent on the p past time series observations preceding it. The model can be represented as:

$$Y_t = c + \sum_{i=1}^{p} \varphi_i Y_{t-i} + \varepsilon_t \tag{4.3.1}$$

where φ_i and c are the model parameters and ε_t is a white noise parameter. In the parking dataset Y_t would correlate to the number of parkings at time t. ARMA model and its variants need to be applied separately for each parking region.

The moving average process MA(q) can be represented as:

$$Y_t = \mu + \varepsilon_t + \sum_{i=1}^{q} \theta_i \, \varepsilon_{t-1} \qquad (4.3.2)$$

where μ is a mean of the MA(q) process and ε_t is again the white noise parameter. Coefficients ε_i and the mean μ represent the model parameters.

Given the definitions of the subprocesses AR(p) and MA(q) we can now define the ARMA(p, q) model as follows:

$$Y_t = c + \varepsilon_t + \sum_{j=1}^{p} \varphi_j Y_{t-j} + \sum_{i=1}^{q} \theta_i \, \varepsilon_{t-1} \qquad (4.3.3)$$

The error terms ε_t are expected to be i.i.d. for all variables, with zero mean and variance σ^2, therefore essentially white noise. [19]

4.3.1. Autoregressive Integrated Moving Average

Autoregressive integrated moving average (ARIMA) is an extension of the ARMA class. ARIMA allows to make the models stationary by differencing the time series before applying the ARMA model to the transformed data. ARIMA can be notated with $ARIMA(p, d, q)$, where d is the order of differencing whereas p and q are the order of AR and MA terms respectively, as described before. The $ARIMA(p, q, d)$ model, with stochastic process $\{X_t\}_{t \in T}$, can be written as follows:

$$\left(1 - \sum_{j=1}^{p} \varphi_j \, L^j\right)(1 - L)^d (Y_t - \mu) = \left(1 + \sum_{i=1}^{q} \theta_i \, L^i\right) \varepsilon_t \qquad (4.3.4)$$

The autoregressive part of the equation is the φ_j parameters, whereas θ_i are the moving average part and ε_t are the error terms. L is the lag operator. It returns the preceding data point $(t - 1)$ of a time series data point at time t. This can be interpreted as a case of $ARMA(p + d, q)$ process that has the autoregressive part with d unit roots. Each difference operation $(1 - L)$ on time series $\{x_t\}_{t \in T}$ removes one suspected unit-root from the process. [4]

A special class, called seasonal autoregressive integrated moving average (SARIMA) can be applied to the traditional ARIMA. SARIMA allows the addition of a seasonal component to the

ARIMA model to help to adjust the model for seasonal effects in the time series. SARIMA can be notated with $SARIMA(p,d,q)(P,D,Q)$ model, where P is the seasonal autoregression term, D is the seasonal order of differencing and Q is the seasonal moving average term. The non-capital (p,d,q) are the non-seasonal counterparts as described before. [20]

4.4. TBATS

Exponential smoothing methods are among the most popular procedures in time series forecasting. However, the methods usually account for only one seasonality in the dataset. The most common models are often based on the additive and multiplicative methods of Holt-Winters [21]. These methods were extended by Taylor [22] to include a second seasonal component:

$$y_t = l_{t-1} + b_{t-1} + s_t^{(1)} + s_t^{(2)} + d_t \tag{4.4.1}$$

$$l_t = l_{t-1} + b_{t-1} + \alpha d_t \tag{4.4.2}$$

$$b_t = b_{t-1} + \beta d_t \tag{4.4.3}$$

$$s_t^{(1)} = s_{t-m_1}^{(1)} + \gamma_1 d_t \tag{4.4.4}$$

$$s_t^{(2)} = s_{t-m_2}^{(2)} + \gamma_2 d_t \tag{4.4.5}$$

Here l_t and b_t are the level and trend components of the time series and $s_t^{(i)}$ represents the ith seasonal component at time t. m_1 and m_2 are the periods of the seasonal cycles and the prediction error is represented by white noise parameter d_t. The initial state variables or "seeds" are represented by $l_0, b_0, \{s_{1-m_1}^{(1)}, \dots, s_0^{(1)}\}$ and $\{s_{1-m_2}^{(2)}, \dots, s_0^{(2)}\}$ and the "smoothing parameters" are denoted as $\alpha, \beta, \gamma_1 \; and \; \gamma_2$.

The model presented above to include a Box-Cox transformation, ARMA errors and *T* seasonal patterns:

$$y_t^{(\omega)} = \begin{cases} \dfrac{(y_t^{\omega} - 1)}{\omega}; & \omega \neq 0 \\ \log y_t & \omega = 0 \end{cases} \tag{4.4.6}$$

$$y_t^{(\omega)} = l_{t-1} + \phi b_{t-1} + \sum_{i=1}^{T} s_{t-m_i}^{(i)} + d_t \tag{4.4.7}$$

$$l_t = l_{t-1} + \phi b_{t-1} + \alpha d_t \tag{4.4.8}$$

$$b_t = (1 - \phi)b + \phi b_{t-1} + \beta d_t \tag{4.4.9}$$

$$s_t^{(i)} = s_{t-m_i}^{(i)} + \gamma_i d_t \tag{4.4.10}$$

$$d_t = \sum_{i=1}^{p} \varphi_i d_{t-i} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + \varepsilon_t \tag{4.4.11}$$

where $\alpha, \beta \; and \; \gamma_i$ are again smoothing parameters for *i*=1, … , *T*. $m_1, \ldots, m_T$ are the seasonal periods, l_t is the local level, b is the long-run trend, b_t is the short-run trend, $s_t^{(i)}$ represents the ith seasonal component, d_t denotes an ARMA(p, q) process and ε_t is a Gaussian white noise parameter with zero mean and constant variance σ^2. $y_t^{(\omega)}$ represents the Box-Cox transformed form of the time series observations with parameter ω where y_t is the observation at time t. In the parking dataset y_t would represent a parking count at time t. TBATS needs to be applied separately for each region.

BATS is an acronym for the key features of the above model: Box-Cox transform, ARMA errors, Trend, and Seasonal components. The model can be complemented with arguments $(\omega, \phi, p, q, m_1, m_2, \ldots, m_T)$ to specify the Box-Cox parameter, damping parameter, ARMA parameters (p, q) and the seasonal periods $m_1, \ldots, m_T$. As an example, the Holt-Winters additive single seasonal method can be denoted as BATS(1, 1, 0, 0, m_1).

TBATS is a Trigonometric form of the BATS model. The difference is that the seasonal components become a trigonometric representation based on Fourier series.

$$s_t^{(i)} = \sum_{j=1}^{k_i} s_{j,t}^{(i)} \tag{4.4.12}$$

$$s_{j,t}^{(i)} = s_{j,t-1}^{(i)} \cos \lambda_j^{(i)} \, s_{j,t-1}^{*(i)} \sin \lambda_j^{(i)} + \gamma_1^{(i)} d_t \tag{4.4.13}$$

$$s_{j,t}^{*(i)} = -s_{j,t-1} \sin \lambda_j^{(i)} + s_{j,t-1}^{*(i)} \cos \lambda_j^{(i)} + \gamma_2^{(i)} d_t \tag{4.4.14}$$

where $\gamma_1^{(i)}$ and $\gamma_2^{(i)}$ are smoothing parameters and $\lambda_j^{(i)} = 2\pi j / m_i$. Moreover, $s_{j,t}^{(i)}$ is the stochastic level of the ith seasonal component and $s_{j,t}^{*(i)}$ is the stochastic growth in the level of the ith seasonal component. k_i denotes the required number of harmonics for the seasonal component with index i. [23]

4.5. Facebook Prophet

FBProphet is an additive regression model, directed at time series forecasting, developed by Facebook. It can also handle non-linear time series and specializes in modeling multiple seasonalities and effects of national holidays. Facebook Prophet operates by decomposing a given time series into three different components: trend g(t), seasonality s(t) and holidays h(t) and an error term ε_t as shown in equation (4.5.1). [24]

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t \tag{4.5.1}$$

When using Facebook Prophet, we have to apply the model separately for each region. In the parking dataset $y(t)$ represents the number of parked cars at time step t.

The trend model g(t) is used for handling the non-periodic component of a time series. There are two options for handling the non-periodicity:

1) Logistic growth model. This model is designed for time series with nonlinear growth. It includes a term for carrying capacity which defines a threshold for the resulting values. For example, the parking regions have a capacity that saturates at the point when all the parking

spots are all taken, and the parking count cannot be a negative value. The logistic model also incorporates S change-points s_j, j=1, 2, …, S that define where the growth rate can change. The rate is adjusted with vector $\boldsymbol{\delta} \in R^S$, where δ_j is the rate change that occurs at time s_j. Total rate at time t is then $k + a(t)^T\delta$ where $\mathbf{a}(t) \in \{0, 1\}^S$.

$$a_j(t) = 1, if\, t \geq s_j$$
$$a_j(t) = 0, otherwise \tag{4.5.2}$$

The logistic growth model takes form:

$$g(t) = \frac{C(t)}{1 + exp\left(-(k + a(t)^T\delta)\left(t - (m + a(t)^T\gamma)\right)\right)} \tag{4.5.3}$$

C is the carrying capacity of the system at different points in time, k is the rate of growth and m is an offset parameter.

To connect the endpoints of segments, the parameter m must be adjusted whenever the rate k changes. The correct adjustment at changepoint t should be computed as:

$$\gamma_t = \left(s_t - m - \sum_{l<t} y_l\right)\left(1 - \frac{k + \sum_{l<t}\delta_l}{k + \sum_{l\leq t}\delta_l}\right) \tag{4.5.4}$$

2) Linear growth model. When forecasting problems that have a linear growth without a carrying capacity. Now the model becomes:

$$g(t) = (k + a(t)^T\delta)t + (m + a(t)^T\gamma) \tag{4.5.5}$$

k is the growth rate, δ defines rate adjustments, m is the offset parameter.

The seasonal model s(t):

$$s(t) = \sum_{n=1}^{N} \left(a_n cos\left(\frac{2\pi nt}{P}\right) + b_n sin\left(\frac{2\pi nt}{P}\right) \right) \quad (4.5.6)$$

where P is the regular period of the seasonality.

Fourier terms are used to model smooth seasonal effects of the time series. Fitting the seasonality requires estimating 2N parameters $\beta = [a_1, b_1, a_2, b_2, \ldots, a_n, b_n]^t$ to the model. This is implemented by combining the seasonality vectors for each value of t in our forecast and historical data into a matrix. Example with weekly seasonality and $N = 3$:

$$X(t) = \left[cos\left(\frac{2\pi(1)t}{7}\right), \ldots, sin\left(\frac{2\pi(3)t}{7}\right) \right] \quad (4.5.7)$$

and formulating the seasonality as follows:

$$s(t) = X(t)\beta \quad (4.5.8)$$

Then a smoothing prior is introduced to the seasonality with $\beta \sim Normal(0, \sigma^2)$.

In many cases holidays and events make an effect on time series. Because national holidays are known information, they add a somewhat predictable signal for the data. Almost all the holidays are visible as a response in the parking dataset. For example, Midsummer and Christmas seem to show a similar effect that weekends have for the count of parkings.

When modeling holidays, the model assumes that effects created by holidays are independent. We assign a parameter κ_i which corresponds the change of each holiday i in the forecast. This is done by generating a matrix of regressors in a similar way to the implementation of seasonality:

$$Z(t) = [1(t \in D_1), \ldots, 1(t \in D_L)] \quad (4.5.9)$$

Where D_i is a yearly set of dates for each holiday i.

As with seasonality, we introduce a prior term $\kappa \sim Normal(0, v^2)$

$$h(t) = Z(t)\kappa \tag{4.5.10}$$

In some cases, the holidays have a window of days around them that have a similar effect as the holiday. To include the window, we introduce additional parameters for the days before and after the holidays, which corresponds the effect of the holiday itself. By adding this window, the surrounding days are essentially treated as holidays as well.

The result is a model that is flexible for time series of different lengths and robust handling of outlier values in the observation points. The model also specializes in modeling of complex seasonalities in time series. It can handle multi-period seasonalities of different periods or even the same period. This means that the model can handle for example the weekends and weekdays separately at a daily level.

4.6. Neural Networks

Inspired by biological neural networks, artificial neural networks (ANN) are a multi-purpose method in machine learning. ANNs have been used to solve various problems for example in image and audio processing, text classification and forecasting.

Conventional machine learning techniques struggle to process raw data in its natural form. Traditionally, building a machine learning system required considerable expertise from machine learning engineers to transfer domain data into machine processable features. It was often required to develop feature extractors to transfer the raw data into feature vectors or another suitable representation. After processing the raw data, it could be processed by the machine learning algorithm, often some kind of classifier, to identify patterns in the input. [25]

Figure 4.6.1 represents a neural network with one hidden layer. It is a so-called feedforward network, that has 3 layers: an input layer, a hidden layer, and an output layer. In this case there is only one hidden layer, but the number of hidden layers n can be any non-negative value ($n \in \mathbb{N}$). The number of input and output neurons can vary according to the application. A network can be described “feedforward” if the outputs of the neurons “travel forward” and are always used as the inputs for the deeper layers, as shown in Figure 4.6.1.

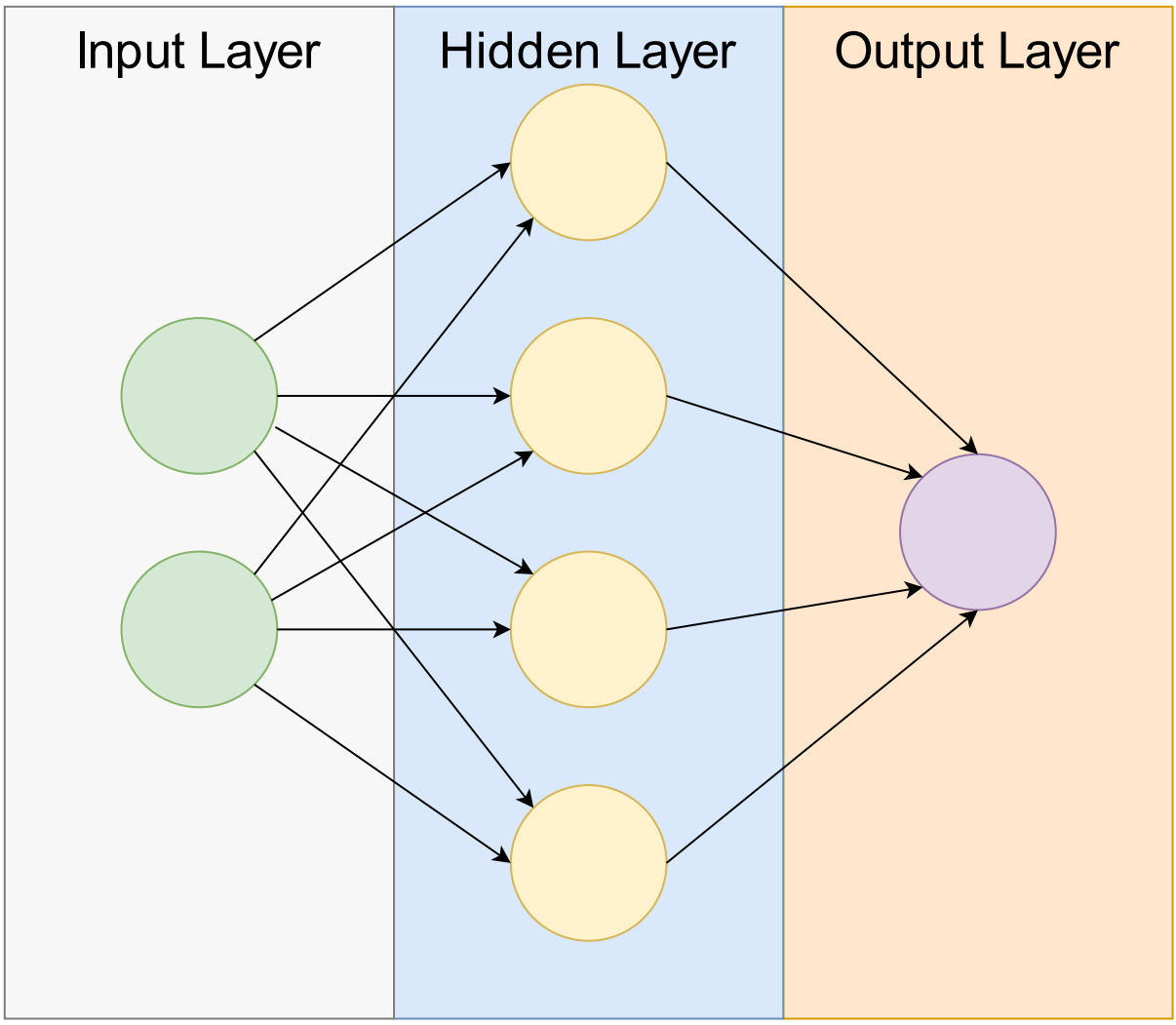

Figure 4.6.1: Example of a feedforward neural network. The circles represent neurons of the neural network and the arrows represent the edges/weights between neurons.

Neural networks consist of artificial neurons that work as connection points that build the network. The artificial neuron receives one or multiple inputs that it then processes to produce an output using an activation function with its weight and bias parameters. There exists a wide selection of activation functions that that can be used for different purposes. The activation function is often considered to be a hyperparameter that can be chosen to fit the purpose and data to be trained. The functionality of an artificial neuron is represented in Figure 4.6.2.

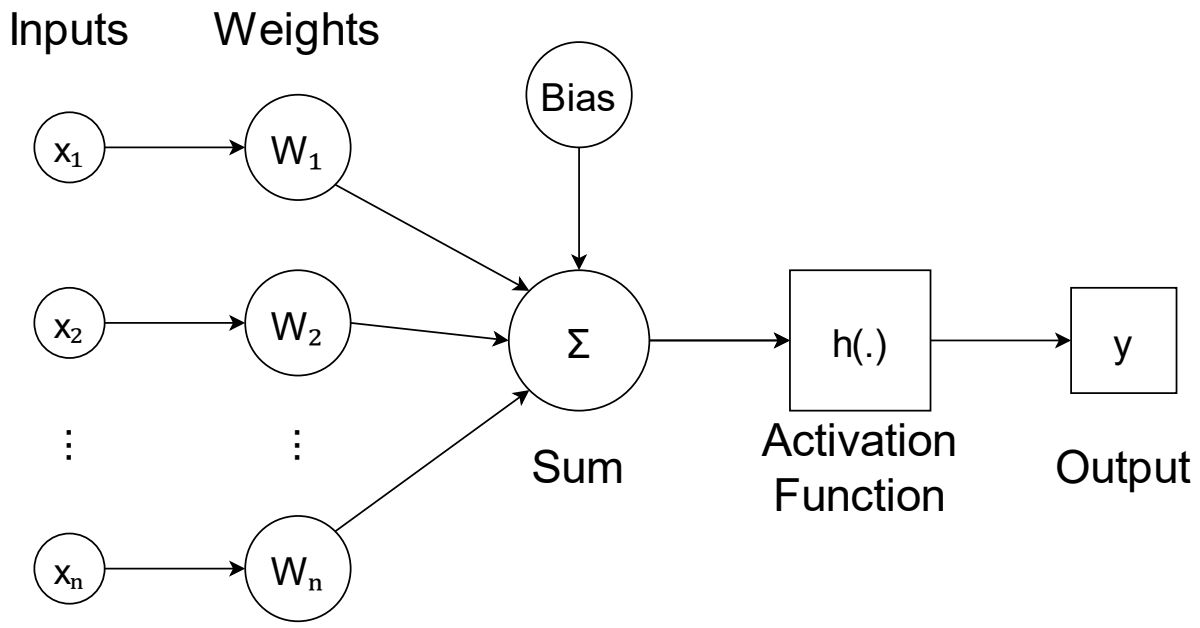

Figure 4.6.2: Structure of an artificial neuron.

4.7. Neural Networks: Long Short-Term Memory

Recurrent neural networks (RNNs) are a class of artificial neural networks, often used for solving problems with data, that can be represented as a sequence of inputs. They can use the outputs of some layers as an input for the previous layer, thus creating cycles in the network. RNNs are an extension of feedforward neural networks (FNN). In an FNN the input signal is fed through the network so that the signal transforms only once per each neuron. However, the RNN might encounter difficulties when dealing with time series problems. The sum of errors from past signals exponentially depends on the magnitude of the weights of the network. This means that backpropagated errors can quickly vanish or "blow up". Therefore, standard RNNs fail to learn data with time lags greater than 5-10 discrete time steps between inputs and target output. Long Short-Term Memory (LSTM) algorithm is more suited to solve these non-trivial time series problems. In this context, a problem is non-trivial when it cannot be solved with random search algorithms.

LSTM consists of memory cells as its basic units. The memory cells include a linear unit with a fixed-weight self-connection. The LSTM units maintain a memory c_t^j at each time step t. The output/activation of the LSTM unit is then:

$$h_t^j = \sigma_t^j \tanh\,(c_t^j) \tag{4.7.1}$$

where σ_t^j is an output gate that modulates the exposure of the memory content. The output gate is calculated by:

$$\sigma_t^j = \sigma(\mathrm{W_o x_t} + U_o \mathrm{h}_{t-1} + V_o \mathrm{c}_t)^j \tag{4.7.2}$$

where V_o is a diagonal matrix and σ is a logistic sigmoid function. W_t is the set of parameters of all the layers of the network at time t. x_t is a data point of the time series at time t. In the parking dataset x_t correlates to the 6 input features.

The memory cell c_t^j is updated by partially forgetting the existing memory and adding a new memory content $\tilde{c}_t^j$

$$c_t^j = \mathrm{f}_t^j c_{t-1}^j + i_t^j \tilde{c}_t^j \tag{4.7.3}$$

where the new memory content is:

$$\tilde{c}_t^j = \tanh\,(W_c \mathrm{x}_t + U_c \mathrm{h}_{t-1})^j \tag{4.7.4}$$

Forget gate f_t^j and input gate i_t^j modulate the ratio to which memory content is forgotten/added to the memory cell. The gates are calculated as follows:

$$\begin{aligned} f_t^j &= \sigma(W_f \mathrm{x}_t + U_f \mathrm{h}_{t-1} + V_f \mathrm{c}_{t-1})^j \\ i_t^j &= \sigma(W_i \mathrm{x}_t + U_i \mathrm{h}_{t-1} + V_i \mathrm{c}_{t-1})^j \end{aligned} \tag{4.7.5}$$

where V_f and V_i are diagonal matrices. [26] We can feed the network all of the regions of the parking dataset at once when fitting it.

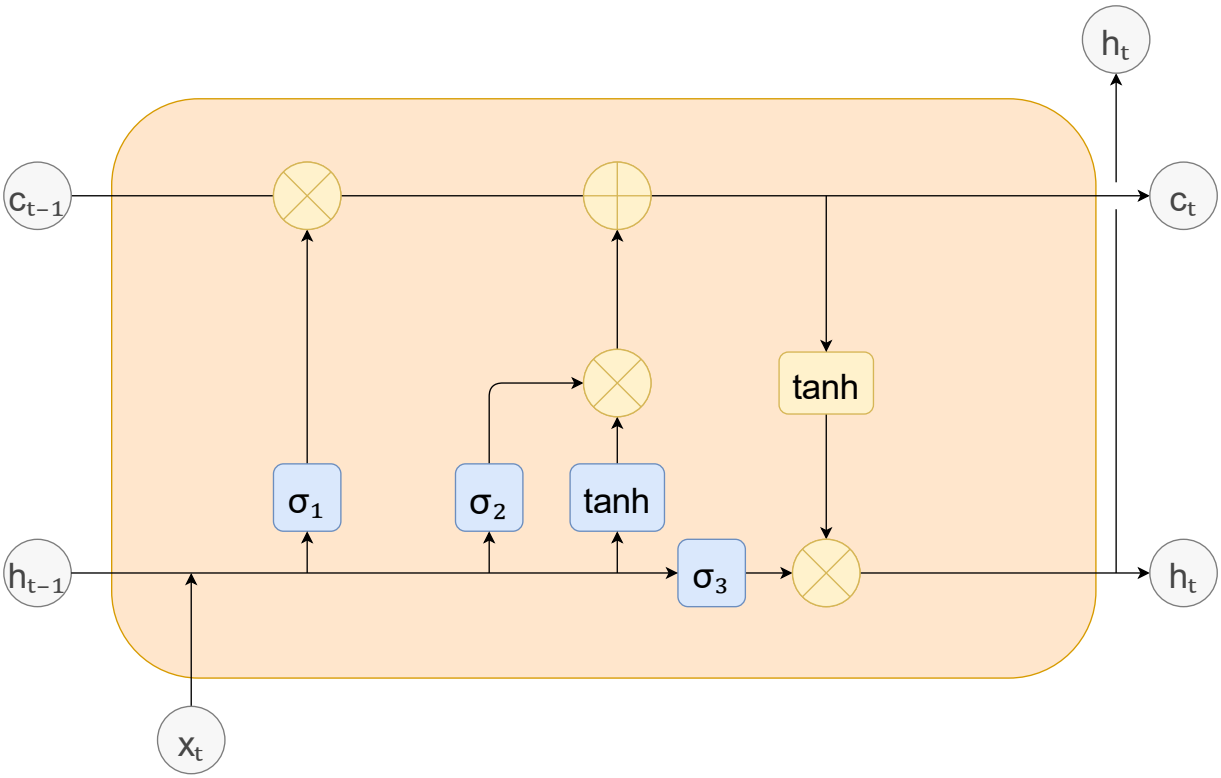

Figure 4.7.1: LSTM layer design.

4.7.1. Backpropagation

To produce the gradients of each layer in an ANN, parameters and the outputs of each layer needs to be combined with non-linear activation functions. The procedure to combine these gradients is often called backpropagation or gradient descent. With backpropagation, the gradient is computed recursively for each layer in the neural network using the chain rule. It performs error propagation in the opposite direction of the data flow by using the errors from the deeper layers to calculate the gradients for the previous layers in the network. Essentially, backpropagation aims to minimize the loss function by adjusting the weights and biases of the network. [27]

Let us denote the cost function as follows $L = loss(s, y)$, where s is the predicted output and y is the actual value, so the number of parked cars at each time step. Now we can describe the chain rule for different parameters as follows. The gradient for a single weight w_{jk}^{l}:

$$\frac{\partial C}{\partial w_{jk}^{l}} = \frac{\partial C}{\partial y_{j}^{l}} * \frac{\partial z_{j}^{l}}{\partial w_{jk}^{l}} \tag{4.7.6}$$

$$y_{j}^{l} = \sum_{k=1}^{m} w_{jk}^{l} a_{k}^{l-1} \tag{4.7.7}$$

$$\frac{\partial y_{j}^{l}}{\partial w_{jk}^{l}} = a_{k}^{l-1} \tag{4.7.8}$$

$$\frac{\partial C}{\partial w_{jk}^{l}} = \frac{\partial C}{\partial y_{j}^{l}} * a_{k}^{l-1} \tag{4.7.9}$$

where j and k are the neuron indices for layers l and $l-1$ and m is the number of neurons in $l-1$ layer. y is the output value and a is the activation value of the neurons.

Now we can similarly describe the gradient for the bias using the chain rule:

$$\frac{\partial C}{\partial b_{j}^{l}} = \frac{\partial C}{\partial y_{j}^{l}} * \frac{\partial z_{j}^{l}}{\partial b_{j}^{l}} \tag{4.7.10}$$

$$\frac{\partial y_{j}^{l}}{\partial b_{j}^{l}} = 1 \tag{4.7.11}$$

$$\frac{\partial C}{\partial b_{j}^{l}} = \frac{\partial C}{\partial y_{j}^{l}} \tag{4.7.12}$$

After all the layers have had the parts of the gradient calculated, the weight and bias parameters can be updated as follows:

$$W = W - \epsilon \frac{\partial C}{\partial W} \tag{4.7.13}$$

$$B = B - \epsilon \frac{\partial C}{\partial B} \tag{4.7.14}$$

where ϵ is the learning rate. W and B are matrix representations of the weights and biases.

4.8. Neural Networks: Gated Recurrent Unit

Gated Recurrent Unit (GRU) was introduced by Chung et al. [28] in order to handle different time scales adaptively. It has similar gating units to LSTM that modulate the data flow inside a unit. Contrary to a LSTM network, GRU networks do not have additional memory cells.

The GRU has an activation h_t^j at time t that is a linear interpolation between the previous activation h_{t-1}^j and the candidate activation $\tilde{h}_t^j$:

$$h_t^j = \left(1 - z_t^j\right) h_{t-1}^j + z_t^j \tilde{h}_t^j \tag{4.8.1}$$

where z_t^j is an update gate, that decides how much the unit updates its activation or content. The update gate formula:

$$z_t^j = \sigma\left(W_z - \mathrm{x}_t \ + U_z \mathrm{h}_{t-1}\right)^j \tag{4.8.2}$$

x_t is a data point of the time series at time t. In the parking dataset x_t correlates to the 6 input features.

GRU takes a linear sum between the current state and the newly calculated state, similarly to LSTM. However, GRU exposes the whole state at each step because it does not have a procedure to control the extent to which its state is exposed.

As with the traditional recurrent unit, candidate activation $\tilde{\mathrm{h}}_t^j$ is calculated with:

$$\tilde{h}_t^j = \tanh\left(W\mathrm{x}_t + U\left(\mathrm{r}_t \odot \mathrm{h}_{t-1}\right)\right)^j \tag{4.8.3}$$

where $\odot$ is an element-wise multiplication operation and r_t is a set of reset gates. If $\mathrm{r}_\mathrm{t}^\mathrm{j}$ gets close to 0, it allows the forget gate to forget the previously computed state, making the unit act as if it is reading the first symbol of an input sequence.

The reset gate $\mathrm{r}_\mathrm{t}^\mathrm{j}$ is computed with:

$$\mathrm{r}_t^j = \sigma(W_r\mathrm{x}_t + U_r\mathrm{h}_{t-1})^j \tag{4.8.4}$$

5. Experiments

To find the optimal model for our problem, a comparison between the potential models needs to be made. In this section we go through the process of applying different models to our problem. All models were implemented using Python 3.6 with the help of the Jupyter Notebook tool.

When choosing a model, we wanted to prioritize a model that can be generalized to all parking regions, meaning there should be no parameters that are unique to only some of the areas. This is because the system is going to grow with more parking regions and more precise data in the future.

As the feature space of the parking dataset we chose the following:

1. **Number of parked cars with one-week lag.**
   Because the forecasts are done for one week in the future, use the one-week lagged values as the feature a.

2. **Hour of the day.**
   $b = 1 \dots 24$

3. **Day of the week.**
   $c = 1 \dots 7$

4. **Day of the year.**
   $d = 1 \dots 366$, including the leap days.

5. **Month number.**
   $e = 1 \dots 12$

6. **Boolean that tells if the timestamp is during a holiday.**
   $f = 1, if\ t \in holidays, else\ f = 0$, where t is the timestamp of the datapoint.

For evaluating the model performance and ranking them against each other, some error metrics are needed. There are a lot of options to choose from and they all have their advantages and disadvantages. For this purpose, we chose Mean Absolute Error (MAE), Root Mean Squared

Error (RMSE) and Mean Absolute Percentage Error (MAPE), because they are popularly used, and robust error metrics.

General steps that were taken with all the models:

1. **Prepare parking data for training.**
   First, we want to prepare the time series for training by clipping outliers from the parking data. The outliers are datapoints with unusually high or low amounts of parkings. We clip these outliers by taking the 0.999 quantile of the data. This seemed to clip the most drastic changes while still keeping the data intact. We also want to resize the dataset values to a smaller scale. Scaling the data makes training the model faster and less prone to blowing up the weight values.

2. **Add Finnish holidays to the data.**
   To add an indication for an external factor, that is holidays, we added a boolean column that indicates, if the datetime stamp for each datapoint is during a Finnish holiday. This is done because the same holiday might be at different dates yearly. With the holiday information the model can adjust itself to the notable variations in parking numbers during the holidays.

3. **Divide dataset into training, validation, and test sets**
   We used training / validation / test set ratio of 50% / 25% / 25%. However, not all models use a validation set. With these models, we use a training / test ratio of 80% / 20%.

   In time series forecasting it is beneficial to select successive datapoints for each of the datasets to keep the seasonal and other effects intact.

4. **Train the model.**
   The model is trained using the pre-evaluated hyperparameters and input data.

5. **Forecast test data.**
   Create forecasts for test set with the chosen model.

6. **Evaluate model performance.**
   Compare forecast with the actual values and calculate MAE, RMSE and MAPE error metrics.

5.1. Linear Regression

The linear regression model is handled very much like the general model. For linear regression we are using the ordinary least squares (OLS) model from the statsmodels python module. The statsmodels implementation of this model needs no hyperparameters to optimize so we just fed the time series data as is for the model.

5.2. Gradient Boosting

For gradient boosting we used the CatBoost library, developed by Yandex. The procedure of constructing this model is mostly the same as in the general steps, but we had to provide some hyperparameters for the model. Furthermore, we need to optimize the parameters for tree depth, number of iterations and learning rate. Because there are quite few hyperparameters to optimize, in comparison to neural networks for instance, it was easy to limit the parameters to optimal range with manual testing. Then the values were optimized using grid search.

5.3. SARIMAX

From the ARMA variants, we want to choose a model that handles our non-stationary data and has multiple long seasonalities.

Before applying the model to the data, we first need to make the data stationary. To make the data stationary, we used the autocorrelation (Figure 3.5) and partial autocorrelation (Figure 3.6) plots to help with the operation. We can see from the plots that there are multiple seasonalities present. The multiple seasonality is tricky to get rid of from the data. Furthermore, the seasonalities are longer in hourly data. For example, the weekly seasonality (24*7=168) would only be 7 in daily time series. The SARIMA model is designed to be used with these shorter seasonalities. In order to deal with the multiple and long seasonalities, external regressors need to be added to the SARIMA model, transforming the model into SARIMAX (SARIMA with exogenous regressors). We addadditional Fourier terms to fit the data:

$$y_t = a + \sum_{i=1}^{M} \sum_{k=1}^{K_i} \left[\alpha \sin\left(\frac{2\pi k t}{p_i}\right) + \beta \cos\left(\frac{2\pi k t}{p_i}\right) \right] + N_t \qquad (5.3.1)$$

The method of adding Fourier terms is flexible in the sense that it allows to add multiple seasons and gives the possibility to adjust the number of the Fourier terms for each seasonality. M is the number of periods, which is in our case 2 (daily 24 and weekly 168).

Each period has its own Fourier series to represent the seasonality. For each of the periods, the number Fourier terms K_i were decided, to match the time series. The number of terms were optimized by minimizing the AIC value of the SARIMAX model.

5.4. TBATS

For the TBATS model, we are using tbats library, designed for time series forecasting. It is based on the paper by Livera et. al. [23]. For the TBATS model, only the seasonal periods need to be provided, in addition to the time series. The seasonal periods have been observed in the parent chapter and as mentioned, we used the daily (24) and weekly (168) periods to train the model.

5.5. Facebook Prophet

FBProphet is a forecasting library developed by Facebook. It is an additive model that has handling for different seasonalities.

The strength of the model is in its ease of use and customizability with seasonalities and external factors. It is easy to add the multiple seasonalities to the model and it is even possible to adjust the Fourier order of the series. The seasonalities are added to the model as Fourier series, similarly to the SARIMAX model. In addition, the model has the possibility to easily add national holidays as an external factor. When added, the model can anticipate the behavior of the time series caused by the holidays.

5.6. Neural Networks

For neural networks, we used the Keras library with the TensorFlow backend, because that is the most commonly used backend. The neural networks include a lot of hyperparameters to optimize, and thus we needed a way to systematically optimize the parameters to get the best results. For the optimization, we utilized the random search tool form Talos library. Talos trains multiple instances of the model with the model parameters randomized. It then compares the

performance of all the trained models and choose the model and hyperparameters with the best performance.

Changes in the steps:

1. **Prepare parking data for training.**
   We scale the dataset values between -1 and 1 because we are using the tanh activation function where the activation values range between -1 and 1.

3.1. Choosing a loss function.

For the loss functions, we chose Mean Squared Error (MSE).

In the beginning of a sequence, the model has not seen many time steps yet and the resulting output might be very inaccurate. We do not want to punish the model for the beginning of the training batches, so we added a warmup period of 50 steps for the beginning of each sequence step, where the losses are not calculated.

3.2. Create parameter dictionary for random search.

Choose a wide variety of hyperparameter values and network depth to get an optimal network model.

Parameters to optimize:

- Number of hidden layers
- Number of neurons in the first layer
- Number of neurons in the hidden layers
- Number of epochs
- Steps per epoch
- Dropout rate

Dropout is a technique that sets a fraction of input units to 0 at each update step during the training of the model. This method is utilized to prevent overfitting. [29] The fraction size is defined by the dropout rate that is set by the user, and in this case, optimized with the talos library.

4. **Train the model.**
   Train the model using the parameter dictionary. Talos uses the hyperparameters in random combinations to produce the optimal model.
   In training we used a batch generator training method where the training data is fed to the model in randomly generated batches. This training method reduces the system

requirements of the training and the batch size can be adjusted according to the GPU and RAM resources available.

4.1. Evaluate the optimal model.

Choose the best model from the Talos trained models by comparing accuracies of the produced models, and thus choosing the best set of hyperparameters.

6. Evaluate model performance.

The warmup period is ignored from accuracy calculations also when evaluating model performance.

6. Results

After applying dataset to all the models, we can use the results from the MAE, RMSE and MAPE error metrics to rank the models against each other. The error metrics are calculated for each parking region separately and then the mean is taken from all regions for each metric. The training set results are shown in Table 6.1. and test set results in Table 6.2.

Training Set	Linear Regression	Gradient Boosting	SARIMAX	TBATS	FBProphet	NN-LSTM	NN-GRU
MAE	9.82	5.94	9.32	x	7.92	4.09	**3.32**
RMSE	13.57	7.97	13.16	x	10.90	5.48	**4.5**
MAPE	50.64%	30.15%	41.44%	x	39.47%	35.63%	**27.32%**

Table 6.1: Mean MAE, RMSE and MAPE error metric values of the different models for the training set. Because TBATS model can only forecast x steps into the future from the training data, it is not possible to forecast the model with the "past" training data and get the training error.

Test Set	Linear Regression	Gradient Boosting	SARIMAX	TBATS	FBProphet	NN-LSTM	NN-GRU
MAE	8.75	**7.40**	8.93	10.10	10.85	7.84	7.53
RMSE	11.85	**9.84**	12.41	13.70	13.55	10.69	10.13
MAPE	44.91%	**38.39%**	41.61%	50.06%	49.92%	42.30%	39.68%

Table 6.2: Mean MAE, RMSE and MAPE error metric values of the different models for the test set.

Figure 6.1, 6.2 and 6.3 show the box plots of the error metrics of test data for each model. Box plot is a method of representing the variance of the error metrics across all parking regions for each model. The plot shows the data divided into 4 quartiles (25% segments), the median and outliers. The blue box extends from Q1 (25th percentile) to Q3 (75th percentile) quartile values of the data, divided by the median Q2 quartile value (50th percentile) represented by the green line in the box. The endpoints of the "whiskers" serve as the minimum and maximum values, excluding the outliers. Outliers are plotted as small circles on the y-axis.

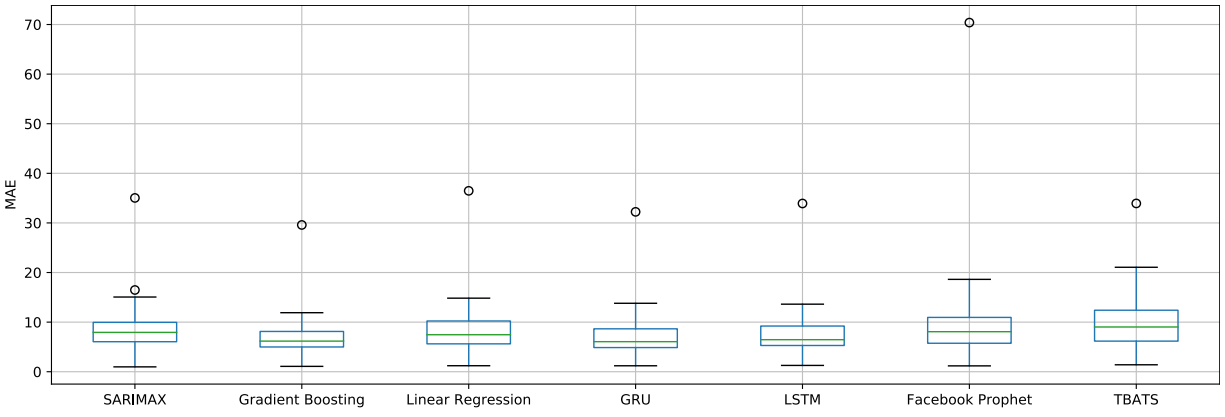

Figure 6.1: Box plot of MAE error metric of test data for each model.

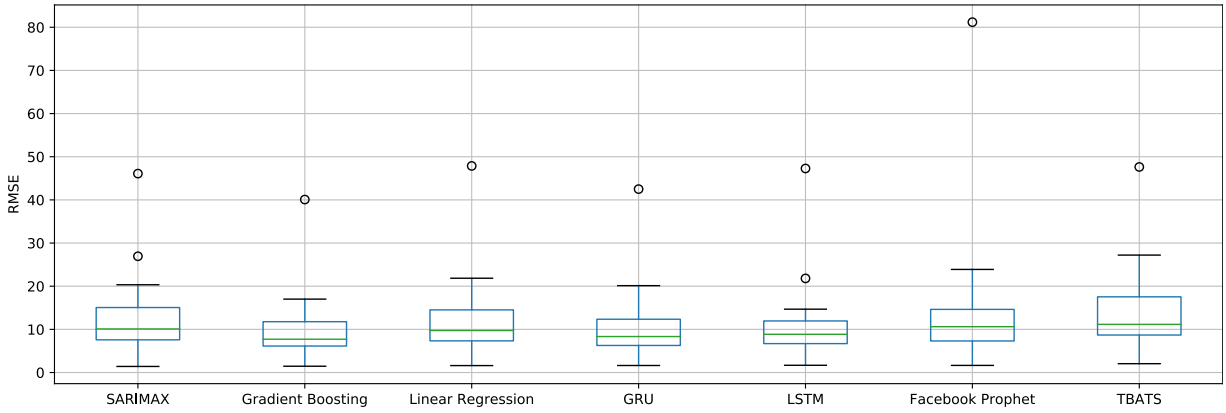

Figure 6.2: Box plot of RMSE error metric of test data for each model.

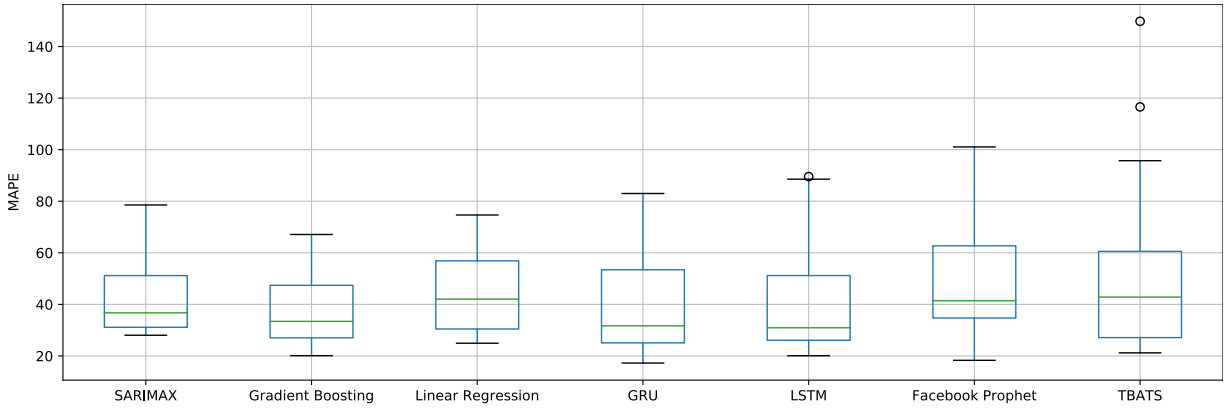

Figure 6.3: Box plot of MAPE error metric of test data for each model.

From the results, it is clear, that Linear Regression averages past values of the time series to create the predictions. The time series plots show that the predictions imitate the actual values with 1-week lag. Linear regression does not have actual predictive power but is used as a baseline for the other models.

FBProphet and TBATS, models that are customized to work well with time series, did not perform well and scored even worse compared to the Linear Regression model. However, they were easy to implement and predicted the values with varying results depending on the parking region. Overall, the results with FBProphet and TBATS were very mixed, and some regions had very good predictions while others had large errors that significantly dragged down the performance.

The SARIMAX model performed moderately but seemed to lack in predicting fluctuating outlier values. For example, when parking numbers on different weekends are varying, the model seems to predict average values for every weekend. This method performs well in regions with stable numbers, but the model is punished in regions where the parking numbers are highly fluctuating. The results show that SARIMAX has an advantage over Linear Regression in the MAPE error metric while the other metrics show similar results.

The best results are provided by the gradient boosting model that shows moderately better performance than the neural network models on all metrics. It has very consistent performance across all parking regions. This is apparent from the box plots as it shows little variance in the error metrics and small outlier values. In addition, the training time is among the fastest of all the tested models. Figures 6.4, 6.5 and 6.6 show the results for region A which had the best predictions of all the regions.

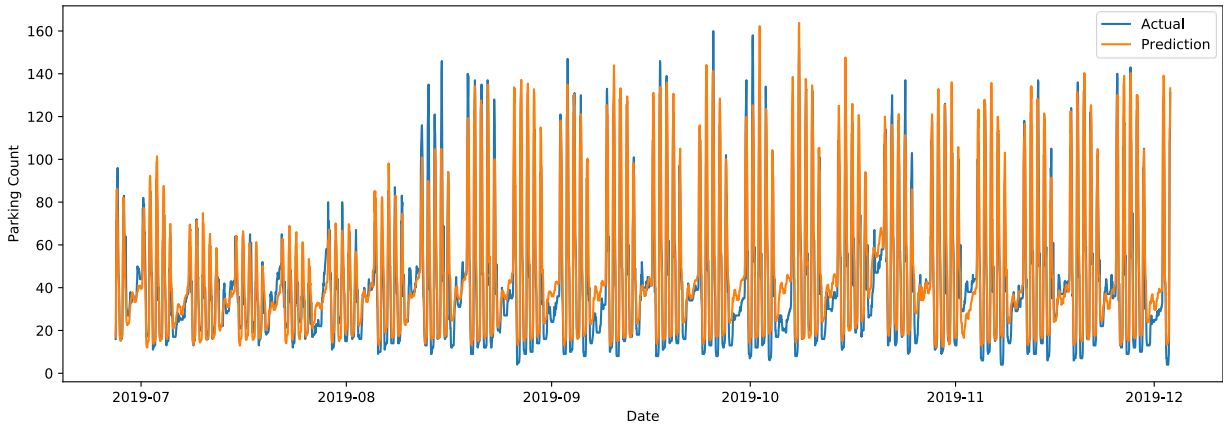

Figure 6.4: A good prediction result for region A. Prediction result created from test set with CatBoost gradient boosting model.

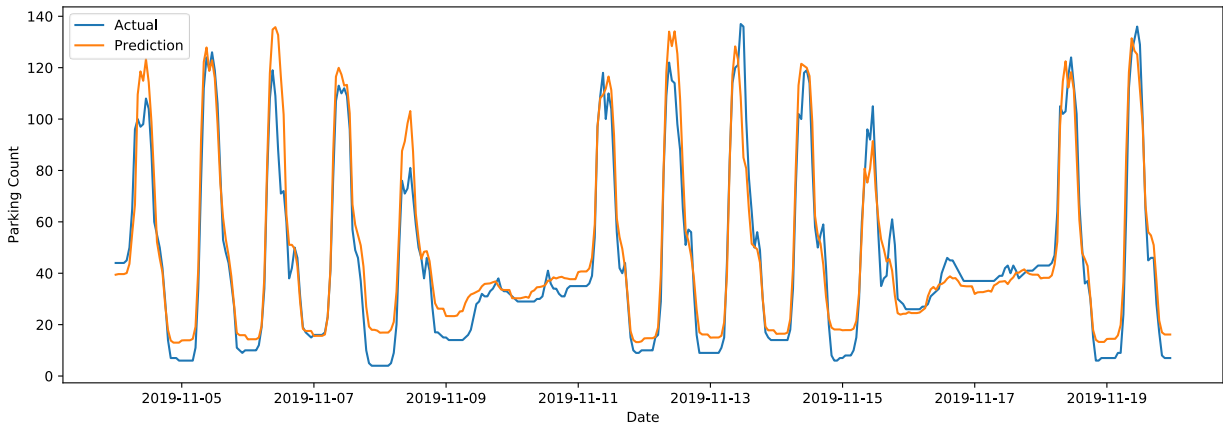

Figure 6.5: A 2-week closeup of the predictions for region A.

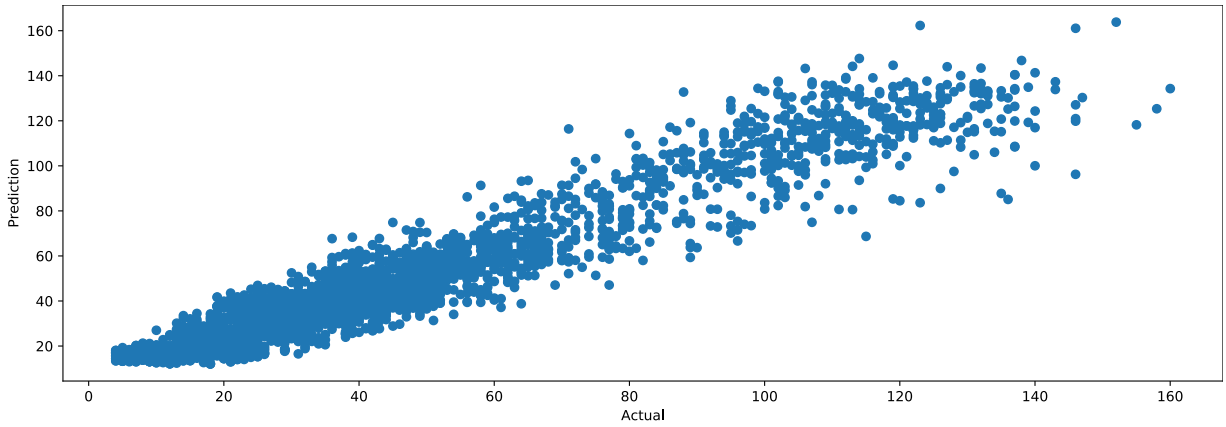

Figure 6.6: A scatter plot of region A. Prediction result created from test set with CatBoost gradient boosting model.

Seems that all the models somewhat struggle with the form of the data, that is semi-cyclic, but has slight fluctuation over time. Even the gradient boosting model did not give optimal results on all the parking regions as seen from the predictions of region B in Figure 6.7.

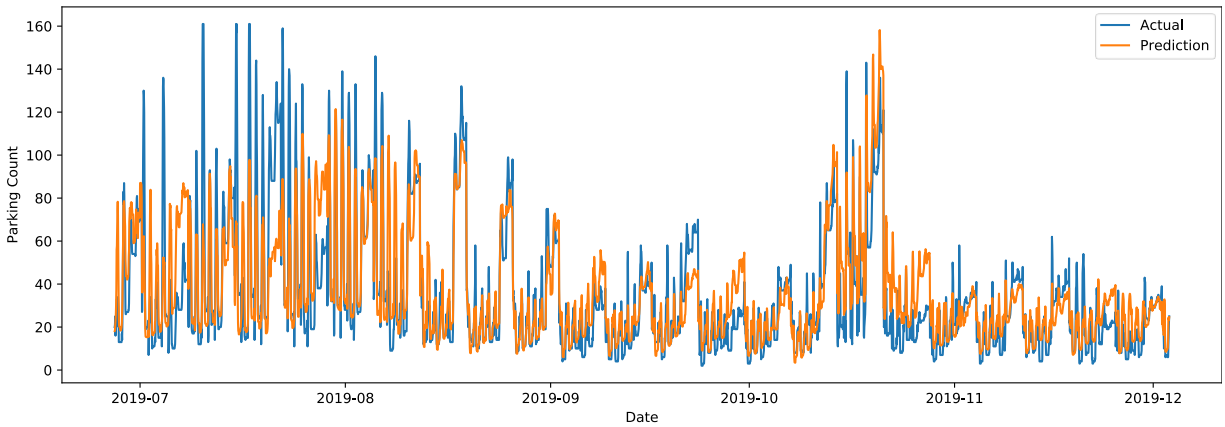

Figure 6.7: A bad prediction result for region B. Prediction result created from test set with CatBoost gradient boosting model.

7. Discussion

When doing initial tests, the parking instances were distributed to streets instead of parking regions. However, the number of parking instances was not so high that the models would be able to give good results on many of the streets. Also, most of the park instances do not have a precise location attached, which makes them unusable for the forecasts. Because of the lacking location data, we decided to create the models on a parking region basis, which would give most of the regions a good model. As the locations for parking instances get more precise in the future, the region-based model will get better predictions and we can adapt the model to work even down to the precision of a street.

When starting the project and gathering different possibilities on how to approach the problem, neural networks seemed like the most promising option. They produce good results in various time series forecasting problems in scientific papers and the industry. However, neural networks have the biggest cost in training times and optimizing the model, according to our experiments. With the available hardware and time in this project, it was very costly to optimize the different neural networks models. We applied different transformations for the dataset, including logarithmic transformation and differentiating the time series with different lags. These transformations did not improve the results drastically. In the early stages of testing, the networks produced the best results and, in the end, ended up ranking quite well. With more time and resources, the neural networks could have been optimized further and might have produced the best results.

While testing the different models in depth, the gradient boosting model started to look very promising, since it was producing similar, and even better results compared to the neural networks. In addition to the performance, the model can be trained in a fraction of the time compared to neural networks. These realizations switched the interest to using the CatBoost model, that has many examples from production use in different industries and our company has also had previous experience with the library.

As we compared the different models, the most interesting factor was the performance of the model, but in addition, the computational resources needed for training and forecasting the model were important as well. Also, it was valuable that developers who are not professionals in machine learning, could be able to use and customize the model when more data and parameters are available for training and forecasting in the future. These criteria match the

CatBoost model well, on the contrary to the Keras model that has a large pool of parameters and documentation to assimilate before it can be used to its full potential.

The production environment where the model is intended to be used, has a limited amount of computational resources available. When selecting a model, it was beneficial to compare the resource requirements so that the model would not hog all the server resources for training. It would be also possible to train the model in an external environment with more resources but for the scope of this project, that would not be practical. When training the model and calculating the forecasts in the production environment, it is possible to automate the process with minimal effort. It is also possible to have close to real time predictions with minimal overhead. By training the models on-server there is no need for external environments or custom setups for the training and forecasting purposes. The CatBoost model is a good candidate for local training, whereas Keras can use a lot of time and resources when there is a lot of data and parameters to optimize.

There are a lot of factors that could affect the number of parkings. Finding the data that is closely correlating with the parking dataset turned out to be quite time consuming and not all desired data is available. For example, event data is quite difficult to find in a centralized API for all of Helsinki. It could be possible to scrape and combine the data from different sources but that would be outside the scope and time constraints of this project. Hopefully, in the future, there will be an API to access event data in Helsinki city area with location and time queries. We believe this kind of information would improve the predictive power of the models and explain some of the outlier numbers.

Weather data is publicly available from the website of Finnish Meteorological Institute (FMI) [30] in a nice format and that was easy to fit in the parking dataset. Particularly, we were interested in the snow levels and temperature, since those are the observations that would affect parking behavior the most. Contrary to expectations, the weather data did not show a clear correlation with the parking data. This is possibly because the parking data is only from recent years and the winters have not been as extreme as before in terms of temperatures and amount of snow. Snow plowing data is also available from Helsinki Region Infoshare (HRI) [31]. Because of the reasons mentioned above, the plowing dataset also showed little to no correlation to the parking time series. Other datasets whose correlation to the parking dataset was tested, include a traffic volumes dataset [32] from HRI and different weather attributes including wind and rain amounts. Ultimately these effects did not show enough correlation to improve the predictive powers of the models.

To train the data, we used a Google Cloud instance [33] to gain performance and saved time. It turned out to be very helpful when optimizing all the different models and trying to find the optimal hyperparameters for each case. It also helped to perform quick tests with external data from different sources.

8. Conclusions

This work described the process of forecasting parking numbers in Helsinki city area. Different models were compared with each other using performance, speed, and ease of customization as the comparison measurement. We introduced the problems of multiple seasonalities and other external factors, like events and holidays that can affect the data.

On the basis of prior research, the original assumption was that neural networks would give the best performance for a parking prediction problem. When conducting the experiments, it turned out that in our application, gradient boosting had better performance and was more efficient in terms of training time. We believe that the performance of the neural networks could be further improved with more available time and resources but that would not be viable in the scope of this project.

The outlines of the forecasting procedure were represented and the process of using the different models was described. The results of the models were presented with the MAE, RMSE and MAPE error metrics as the main factor in selecting the best performing model. Though performance was the priority, other factors were considered as well, including training time and resource requirements.

For future work on this feature, it would be beneficial to include further external factors to the data. For example, public event data would be helpful, so that we can predict the parking number fluctuations on an area basis. Furthermore, it would be optimal if the event data would be available from a centralized API where one could fetch them with datetime and geographic area. Additionally, in the future, the datapoints will have a more precise location, and the number of datapoints will increase. These improvements will help to increase the prediction abilities of the models and enable the possibility to create predictions even down to the precision of a street.

Bibliography

[1] D. Shoup, ‘Free Parking or Free Markets’, in *Parking and the City*, 1st ed., D. Shoup, Ed. First edition. | New York, NY : Routledge, 2018.: Routledge, 2018, pp. 270–275.

[2] A. Jung, ‘Machine Learning: Basic Principles’, *arXiv:1805.05052 [cs, stat]*, May 2019, Accessed: Nov. 12, 2019. [Online]. Available: http://arxiv.org/abs/1805.05052.

[3] G. Bontempi, S. Ben Taieb, and Y.-A. Le Borgne, ‘Machine Learning Strategies for Time Series Forecasting’, in *Business Intelligence*, vol. 138, M.-A. Aufaure and E. Zimányi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 62–77.

[4] P. J. Brockwell and R. A. Davis, *Introduction to Time Series and Forecasting*. Springer, 2016.

[5] C. Chatfield, *Time-Series Forecasting*. CRC Press, 2000.

[6] A. Ionita, A. Pomp, M. Cochez, T. Meisen, and S. Decker, ‘Where to Park?: Predicting Free Parking Spots in Unmonitored City Areas’, Jun. 2018, pp. 1–12, doi: 10.1145/3227609.3227648.

[7] B. Xu, O. Wolfson, J. Yang, L. Stenneth, P. S. Yu, and P. C. Nelson, ‘Real-Time Street Parking Availability Estimation’, in *2013 IEEE 14th International Conference on Mobile Data Management*, Jun. 2013, vol. 1, pp. 16–25, doi: 10.1109/MDM.2013.12.

[8] Z. Chen, J. (Cecilia) Xia, and B. Irawan, ‘Development of Fuzzy Logic Forecast Models for Location-Based Parking Finding Services’, *Mathematical Problems in Engineering*, 2013. https://www.hindawi.com/journals/mpe/2013/473471/ (accessed Nov. 21, 2019).

[9] A. Koster, A. Oliveira, O. Volpato, V. Delvequio, and F. Koch, ‘Recognition and Recommendation of Parking Places’, in *Advances in Artificial Intelligence -- IBERAMIA 2014*, Cham, 2014, pp. 675–685, doi: 10.1007/978-3-319-12027-0_54.

[10] A. Nandugudi, T. Ki, C. Nuessle, and G. Challen, ‘PocketParker: Pocketsourcing parking lot availability’, *UbiComp 2014 - Proceedings of the 2014 ACM International*

Joint Conference on Pervasive and Ubiquitous Computing, pp. 963–973, Sep. 2014, doi: 10.1145/2632048.2632098.

[11] S. Mathur *et al.*, 'ParkNet: Drive-by Sensing of Road-Side Parking Statistics', presented at the MobiSys'10 - Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services, Aug. 2010, pp. 123–136, doi: 10.1145/1814433.1814448.

[12] T. Tiedemann, T. Voegele, M. M. Krell, J. H. Metzen, and F. Kirchner, 'Concept of a Data Thread Based Parking Space Occupancy Prediction in a Berlin Pilot Region', in *AAAI Workshop: AI for Transportation*, 2015.

[13] R. Hössinger *et al.*, 'Development of a Real-Time Model of the Occupancy of Short-Term Parking Zones', *Int. J. ITS Res.*, vol. 12, no. 2, pp. 37–47, May 2014, doi: 10.1007/s13177-013-0069-5.

[14] C. Badii, P. Nesi, and I. Paoli, 'Predicting Available Parking Slots on Critical and Regular Services by Exploiting a Range of Open Data', *IEEE Access*, vol. 6, pp. 44059–44071, 2018, doi: 10.1109/ACCESS.2018.2864157.

[15] R. H. Myers, *Classical and Modern Regression with Applications*. PWS-KENT, 1990.

[16] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, 'CatBoost: unbiased boosting with categorical features', *arXiv:1706.09516 [cs]*, Jan. 2019, Accessed: Mar. 26, 2020. [Online]. Available: http://arxiv.org/abs/1706.09516.

[17] Q. Wu, C. J. C. Burges, K. M. Svore, and J. Gao, 'Adapting boosting for information retrieval measures', *Inf Retrieval*, vol. 13, no. 3, pp. 254–270, Jun. 2010, doi: 10.1007/s10791-009-9112-1.

[18] Y. Wang, 'Prediction of weather impacted airport capacity using ensemble learning', in *2011 IEEE/AIAA 30th Digital Avionics Systems Conference*, Seattle, WA, USA, Oct. 2011, pp. 2D6-1-2D6-11, doi: 10.1109/DASC.2011.6096002.

[19] B. E. Hansen and Hamilton, James D., 'TIME SERIES ANALYSIS', *Econometric Theory*, vol. 11, no. 3, pp. 625–630, Jun. 1995, doi: 10.1017/S0266466600009440.

[20] W. Hu, S. Tong, K. Mengersen, and D. Connell, 'Weather Variability and the Incidence of Cryptosporidiosis: Comparison of Time Series Poisson Regression and SARIMA

Models', *Annals of Epidemiology*, vol. 17, no. 9, pp. 679–688, Sep. 2007, doi: 10.1016/j.annepidem.2007.03.020.

[21] P. S. Kalekar, 'Time series Forecasting using Holt-Winters Exponential Smoothing', p. 13.

[22] J. W. Taylor, 'Short-term electricity demand forecasting using double seasonal exponential smoothing', *Journal of the Operational Research Society*, vol. 54, no. 8, pp. 799–805, Aug. 2003, doi: 10.1057/palgrave.jors.2601589.

[23] A. M. De Livera, R. J. Hyndman, and R. D. Snyder, 'Forecasting Time Series With Complex Seasonal Patterns Using Exponential Smoothing', *Journal of the American Statistical Association*, vol. 106, no. 496, pp. 1513–1527, Dec. 2011, doi: 10.1198/jasa.2011.tm09771.

[24] S. J. Taylor and B. Letham, 'Forecasting at scale', PeerJ Inc., e3190v2, Sep. 2017. doi: 10.7287/peerj.preprints.3190v2.

[25] Y. LeCun, Y. Bengio, and G. Hinton, 'Deep learning', *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.

[26] T. Guo, Z. Xu, X. Yao, H. Chen, K. Aberer, and K. Funaya, 'Robust Online Time Series Prediction with Recurrent Neural Networks', in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, Oct. 2016, pp. 816–825, doi: 10.1109/DSAA.2016.92.

[27] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, 'Learning representations by back-propagating errors', *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986, doi: 10.1038/323533a0.

[28] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, 'Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling', *arXiv:1412.3555 [cs]*, Dec. 2014, Accessed: Dec. 16, 2019. [Online]. Available: http://arxiv.org/abs/1412.3555.

[29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting', p. 30.

[30] 'Open data - Finnish Meteorological Institute'. https://en.ilmatieteenlaitos.fi/open-data (accessed Apr. 30, 2020).

[31] ‘Stara Snowplow API - Helsinki Region Infoshare’. https://hri.fi/data/en_GB/dataset/rajapinta-staran-lumiaurojen-sijantitietoihin (accessed Apr. 30, 2020).

[32] ‘Traffic Volumes in Helsinki - Helsinki Region Infoshare’. https://hri.fi/data/en_GB/dataset/liikennemaarat-helsingissa (accessed May 05, 2020).

[33] ‘Cloud AI’, *Google Cloud*. https://cloud.google.com/products/ai?hl=fi (accessed Apr. 30, 2020).